Setup Procedures: Steps to Follow

This appendix first takes us through some pre-setup checks that we need to do, to ensure that the tables we want to replicate are suited for Q replication. Next, we go through various scenarios and give step-by-step instructions. If a command in a scenario is the same as in a previous scenario, then it is not repeated, but a reference is made to the previous command.

Tools available to help set up Q replication

This section looks at the tools available to help set up the three layers of Q replication.

The database layer

The DB2 Control Center or Optim Data Studio can be used to create the databases, enable archive logging, and create the source/target tables.

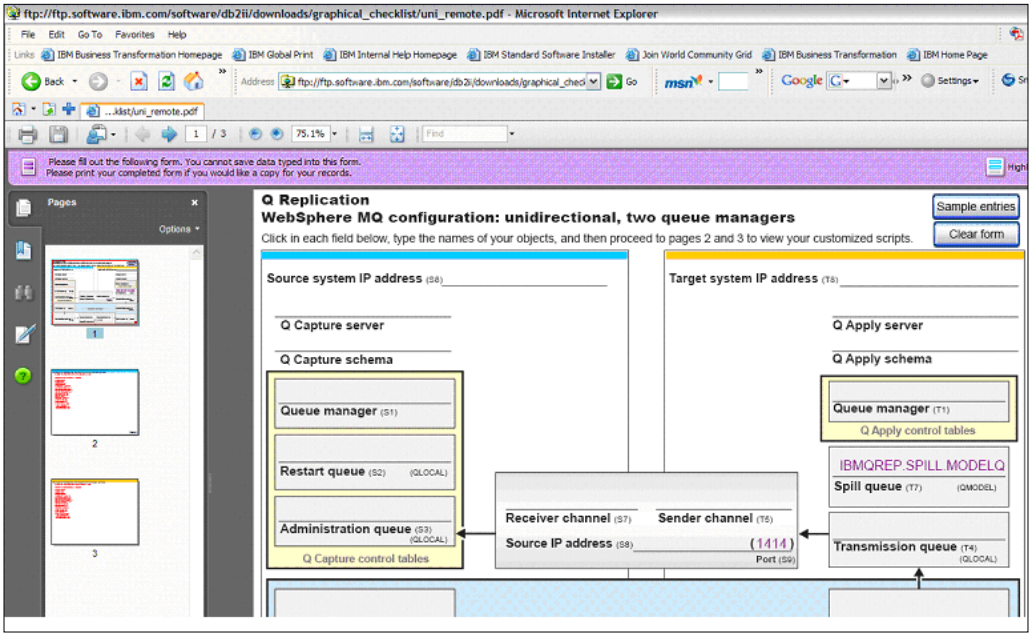
The WebSphere MQ layer

There are a couple of aids to setting up the WebSphere MQ layer.

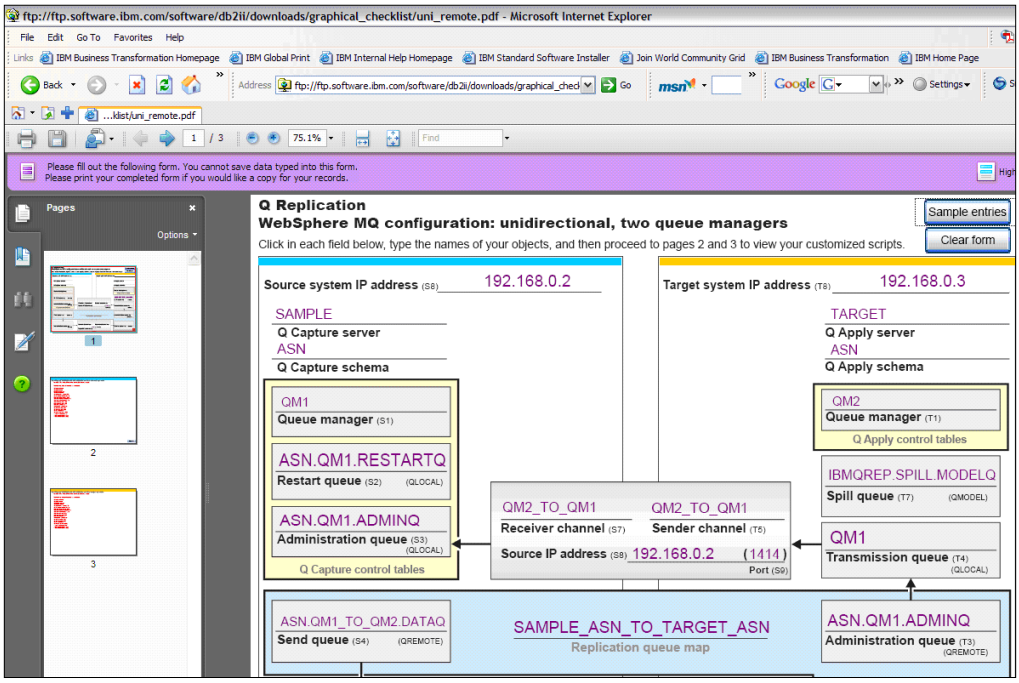
Firstly, there is an article on the IBM developerWorks website called *Graphical tool for generating WebSphere MQ setup scripts for Q replication and event publishing*, which describes a free downloadable tool. This tool generates the MQ script that we need to define the queues from a screen where we type in the queue names. There is also a **Sample entries** button, which will give us some default queue names if we are unsure of the names we want to use. The tool is a series of four PDF files, which we can open and print but cannot save. The four scenarios covered are:

- Unidirectional replication (two Queue Managers)
- Bidirectional or peer-to-peer (P2P) replication (two Queue Managers)
- Event publishing (two Queue Managers)
- Event publishing (one Queue Manager)

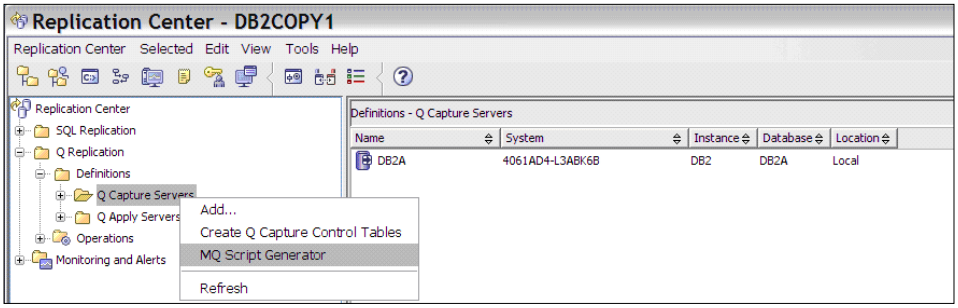
The following screenshot shows the PDF file for unidirectional replication:



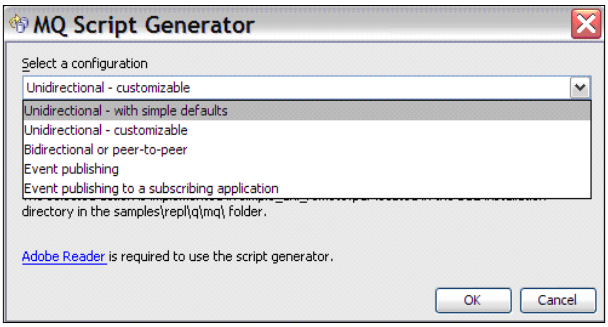
And if we click the **Sample entries** button, the queue names are filled in for us.



Secondly, for DB2 9.7 onwards, there is an **MQ Script Generator** built into the **Replication Center**, which can be accessed by right-clicking on the **Q Capture Servers** folder and selecting **MQ Script Generator** as shown in the following screenshot:



We have a choice of five possible scenarios, as shown in the following screenshot:



This opens up a PDF file, and we have to follow the instructions in there.

The Q replication layer

We can use ASNCLP commands to set up the Q replication layer. We must not forget the Replication Center when it comes to setting up the Q replication layer. From here, we can create the control tables, create Replication Queue Maps, and Q subscriptions and administer Q Capture and Q Apply. There is also the possibility of writing SQL to set up the Q replication layer, but this should be avoided whenever possible. Use the Replication Center the first time you are setting up replication, and once you are happy with the principles use ASNCLP commands.

Pre-setup evaluation

There are certain issues that need to be considered when planning a Q replication scenario to make sure that there are no surprises at the end of the process. Following is a checklist of points that we have come across and in what situations they are an issue. All candidate tables should be checked for:

- Referential integrity
- Triggers
- Identity columns
- Sequence objects
- Absence of unique key
- Applications processing different rows in the same table

Let's look at each of these in more detail.

Referential integrity

Replication scenario: All.

All source tables that are linked through referential integrity must use the same Replication Queue Map.

Triggers

Replication scenario: Bidirectional, P2P.

We need to analyze any trigger that is on a source table to understand which tables are touched by the trigger and what is the impact of the trigger's action. Then we can determine if we should:

- Leave it alone as is
- Remove it
- Drop and recreate it to reflect replication's behavior and continue to have the correct impact on the data

If we need to modify the trigger, it is usually to validate if the update comes from Q Apply or from the local application. If it comes from Q Apply then we do not want to fire the trigger. We can validate this with SQL using the `CURRENT USER` special registry parameter. Say the user ID used to run Q Apply is `QREPADM`, then we can use the `WHEN` portion of the trigger's syntax to only fire if `CURRENT USER NOT = 'QREPADM'`.

Identity columns

Identity columns and Sequence objects (see next topic) can be problematic when we bring replication into the mix. These objects are automated in sequence or through identity generators. This functionality is commonly used in order to relieve the application from the responsibility of maintaining the next value, next value plus one, and so on.

The definition of identity columns, can include a start (or seed), an increment (skip how many numbers), and a maximum value. The maximum is usually way out of reach. Other characteristics include whether these values will always be generated by DB2 or if the user/application can provide a number.

We have two options for identity columns: `GENERATED ALWAYS AS IDENTITY` or `GENERATED BY DEFAULT AS IDENTITY`.

The GENERATED ALWAYS AS IDENTITY clause causes a problem and will result in the replication definition process (ASNCLP or Replication Center) to fail. The message is misleading and mentions missing target column map. The real problem is that with bidirectional replication the value will sometimes come from the other system and be delivered by Q Apply. If the definition of the identity column is ALWAYS then Q Apply could not insert the complete row with the same values as the source.

An example of a problematic identity column (inv_num) definition would be:

```
CREATE TABLE hmtab
(inv_num INT NOT NULL UNIQUE GENERATED ALWAYS AS IDENTITY
(START WITH 20, INCREMENT BY 1, NO CACHE),
item CHAR (100))
```

We can use the following query to check if we have any tables with identity columns which were defined as GENERATED ALWAYS AS IDENTITY:

```
db2 "SELECT SUBSTR(tabschema,1,10) AS tabschema, SUBSTR(tabname,1,10) AS
tabname, identity, generated FROM syscat.columns WHERE identity = 'Y' AND
generated = 'A' "
```

The identity column can contain the following values:

- N: Not an identity column.
- Y: Identity column.

The generated column can contain the following values:

- A: Column value is always generated.
- D: Column value is generated by default.
- Blank: Column is not generated.

Any such tables need to be altered to GENERATED BY DEFAULT AS IDENTITY as follows:

```
ALTER TABLE <table-name> ALTER COLUMN <column-name> SET GENERATED BY
DEFAULT
```

Altering tables to be GENERATED BY DEFAULT AS IDENTITY will have no immediate impact on any of the applications behavior and will continue to generate values (in the right order). However, it will enable Q Apply to successfully bring over the entire newly inserted row at the source including the identity column value from the source.

We do not have to worry about the cache setting, which can be:
CACHE <n> or NO CACHE.

Sequence objects

The `SEQUENCE` function allows us to define a counter which is not dependent on any particular column in a table, but is defined for the database in which it is created. This is different from an identity column, which is table/column dependent. We can specify various options when we define the sequence—a start value, an increment value, and whether we want values cached by DB2 or not (or accept the defaults). We can also specify a maximum value and a minimum value, and whether we want to cycle back to the beginning when we hit the maximum/minimum value.

Following line shows an example of creating a sequence (called `hm`):

```
CREATE SEQUENCE hm AS INTEGER START WITH 10 INCREMENT BY 2
```

Replication relies heavily upon the presence of a unique or primary key on each table involved in replication. This key column is used by Q Apply to correctly match updated rows from the source to target.

We try to avoid scenarios where applications at either site might produce duplicate keys as this would cause an artificial conflict, but still result in data loss.

A common strategy is to seed each server with a different value and then skip values in a way to avoid the same number on any server. For example with two servers—`SY1` and `SY2`:

- `SY1`: Start with `0` increment by `2`.
- `SY2`: Start with `1` increment by `2`.

The result is that `SY1` will have all even values, and `SY2` will have all odd values.

With three servers—`SY1`, `SY2`, and `SY3`, the starting and incremental values are:

- `SY1`: Start with `0` increment by `3`.
- `SY2`: Start with `1` increment by `3`.
- `SY3`: Start with `2` increment by `3`.

Absence of a unique key

If we do not have a primary key, unique constraint, or unique index on the source table, then Q Apply will automatically create a unique index for the new target table that is based on all valid, subscribed source columns.

If we have not specified a method for establishing uniqueness at the target, then Q Apply uses a primary key, unique constraint, or unique index at the target to enforce the uniqueness of each row when it applies the row to target tables or parameters in stored procedures.

Replicating views

We cannot replicate views using Q replication.

Applications processing different rows in the same table

Replication scenario: Bidirectional.

In a bidirectional scenario, we need to evaluate whether conflicts are possible from an application point of view. This is covered in the *The different types of Q replication – Bidirectional replication* section of *Chapter 1, Q Replication Overview*, which discussed the two different types of bidirectional replication, one where there is an active/passive setup and the second type where applications are updating both sides of the bidirectional setup at the same time—an active/active setup.

First steps—common to all scenarios

This section details the common first steps for all scenarios.

Database creation

In each scenario, we will use either one, two, three, or four databases called `DB2A`, `DB2B`, `DB2C`, and `DB2D`. In all scenarios the required number of databases should be created as shown next. The instructions in each scenario specify how many databases to create.

The user id used to set up replication must have write authority to a directory called `c:\temp`, which is the directory pointed to by the DB2 `BACKUP` command.

In the following sections, we talk about *source* and *target* databases. The difference between the two is that source databases have archive logging switched on, whereas this is not required for target databases.

A source database (`DB2A`) can be created using the DB2 `SYSA_crt_db2a.sql` script file:

```
CONNECT RESET;
DROP DB db2a;
CREATE DB db2a;
CONNECT TO db2a;
CREATE TABLE eric.t1(c1 INT NOT NULL, c2 INT, c3 CHAR(10));
ALTER TABLE eric.t2 ADD CONSTRAINT t1p PRIMARY KEY (c1);
```

```
CREATE TABLE eric.t2(c1 INT NOT NULL, c2 INT, c3 CHAR(10));
ALTER TABLE eric.t2 ADD CONSTRAINT t2p PRIMARY KEY (c1);

CREATE TABLE eric.t3(c1 INT NOT NULL, c2 INT, c3 CHAR(10));
ALTER TABLE eric.t3 ADD CONSTRAINT t3p PRIMARY KEY (c1);

CONNECT RESET;
UPDATE DB CFG FOR db2a USING logarchmeth1 disk:c:\temp;
BACKUP DATABASE db2a TO c:\temp;
```

From CLP-A, run the file as:

```
$ db2 -tvf SYSA_crt_db2a.sql
```

A target database (`DB2B`) can be created using the DB2 `SYSB_crt_db2b.sql` script file (note that we are switching on archive logging, but this is not necessary):

```
CONNECT RESET;
DROP DB db2b;
CREATE DB db2b;
UPDATE DB CFG FOR db2b USING logarchmeth1 disk:c:\temp;
BACKUP DATABASE db2b TO c:\temp;
```

From CLP-B, run the file as:

```
$ db2 -tvf SYSB_crt_db2b.sql
```

We found that for P2P four-way replication, depending on the value of the database manager configuration parameter `maxagents`, we may have to increase it. We increased it from the default value of 200 to 2000 (otherwise we might get `SQL1226N` errors).

For each database that we create, we need to open a DB2 CLP session.

Setting up a password file

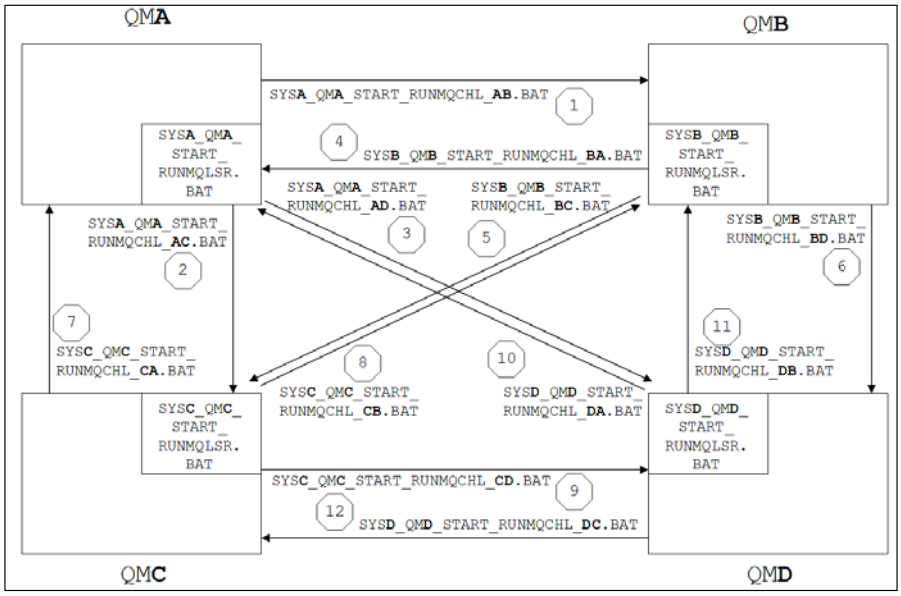
If we are setting up replication between databases on different machines, then we need to set up a password file on each Q Apply server. Refer to the *The password file* section of *Chapter 6*, *Administration Tasks*, for details on how to initialize and populate a password file.

Queue Manager processing

Depending on the scenario, the Queue Managers `QMA`, `QMB`, `QMC`, and `QMD` should be created. Refer to the *Create/start/stop a Queue Manager* section of *Chapter 4*, *WebSphere MQ for the DBA*, for a description of how to create a Queue Manager.

Listeners and Channels

Consider the following diagram, which shows a P2P four-way setup. Each Queue Manager in a communicating pair will have one Listener, and there will be a Channel *to* the Queue Manager and a Channel *from* the Queue Manager to the other Queue Manager in the pair. It shows the start Listener batch file names and the start Channel batch file names for all four Queue Managers which are used in this appendix.



Starting Q Capture and Q Apply

To start Q Capture and Q Apply, follow the instructions in the *Q Capture administration – Starting Q Capture, Starting Q Apply* sections of *Chapter 6*.

Text file creation for the amqsput command

To test the WebSphere layer, we need to create various text files, which will be used as input files to the `amqsput` command. For each filename, populate the file with the corresponding `test<x>` word. So for example, `SYSA_QMA_TEST1.TXT` file will contain `test1` and `SYSB_QMB_TEST2.TXT` will contain `test2` and so on.

Contents of text files for the `amqsput` command

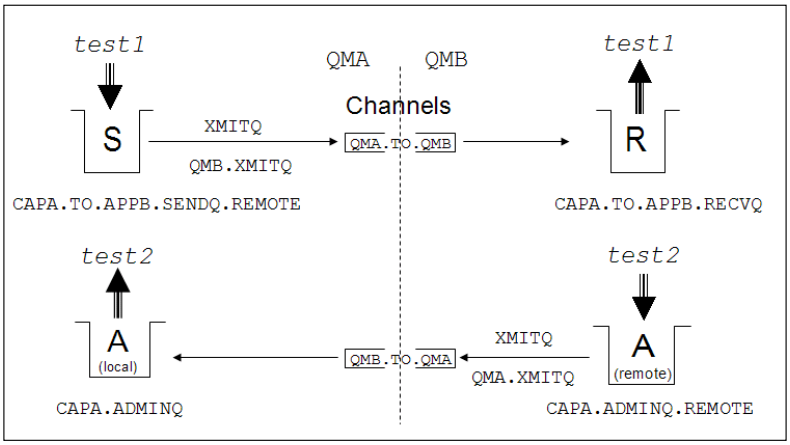
File	Uni	EP	Bi	P2P2W	P2P3W	P2P4W
SYSA_QMA_TEST1.TXT	X		X	X	X	X
SYSA_QMA_TEST2.TXT			X	X	X	X
SYSA_QMA_TEST3.TXT					X	X
SYSA_QMA_TEST4.TXT					X	X
SYSA_QMA_TEST5.TXT						X
SYSA_QMA_TEST6.TXT						X
SYSB_QMB_TEST2.TXT	X					
SYSB_QMB_TEST4.TXT			X	X		
SYSB_QMB_TEST3.TXT			X	X		
SYSB_QMB_TEST5.TXT					X	
SYSB_QMB_TEST6.TXT					X	
SYSB_QMB_TEST7.TXT					X	X
SYSB_QMB_TEST8.TXT					X	X
SYSB_QMB_TEST9.TXT						X
SYSB_QMB_TEST10.TXT						X
SYSB_QMB_TEST11.TXT						X
SYSB_QMB_TEST12.TXT						X
SYSC_QMC_TEST9.TXT					X	
SYSC_QMC_TEST10.TXT					X	
SYSC_QMC_TEST11.TXT					X	
SYSC_QMC_TEST12.TXT					X	
SYSC_QMC_TEST13.TXT						X
SYSC_QMC_TEST14.TXT						X
SYSC_QMC_TEST15.TXT						X
SYSC_QMC_TEST16.TXT						X
SYSC_QMC_TEST17.TXT						X
SYSC_QMC_TEST18.TXT						X
SYSD_QMD_TEST19.TXT						X

File	Uni	EP	Bi	P2P2W	P2P3W	P2P4W
SYSD_QMD_TEST20.TXT						X
SYSD_QMD_TEST21.TXT						X
SYSD_QMD_TEST22.TXT						X
SYSD_QMD_TEST23.TXT						X
SYSD_QMD_TEST24.TXT						X

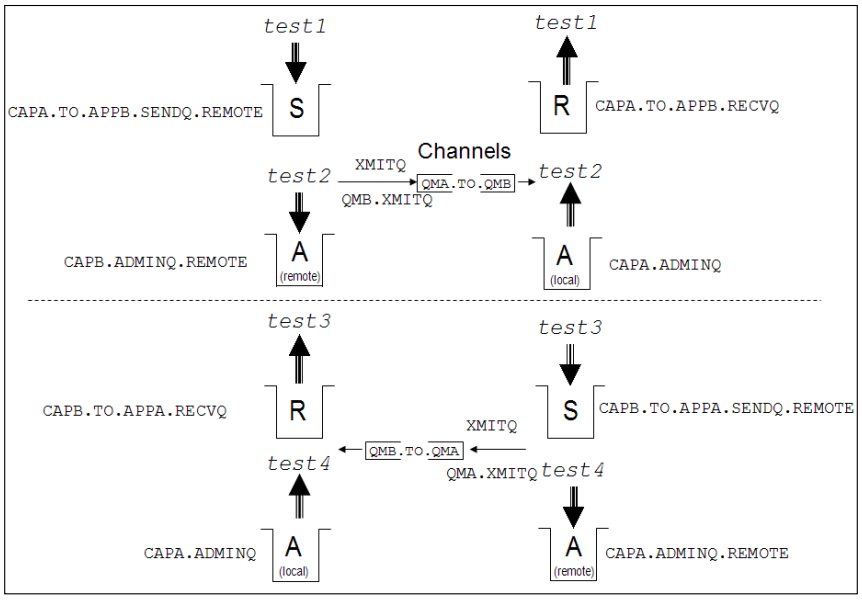
We will put the contents of these files onto the appropriate Send Queues. The `amqsput` command was discussed in the *WebSphere MQ sample programs – server* section of *Chapter 4*. In the scenarios ,we show Windows examples of the batch files.

The following diagram shows the Queues and text files for two scenarios.

For unidirectional replication:



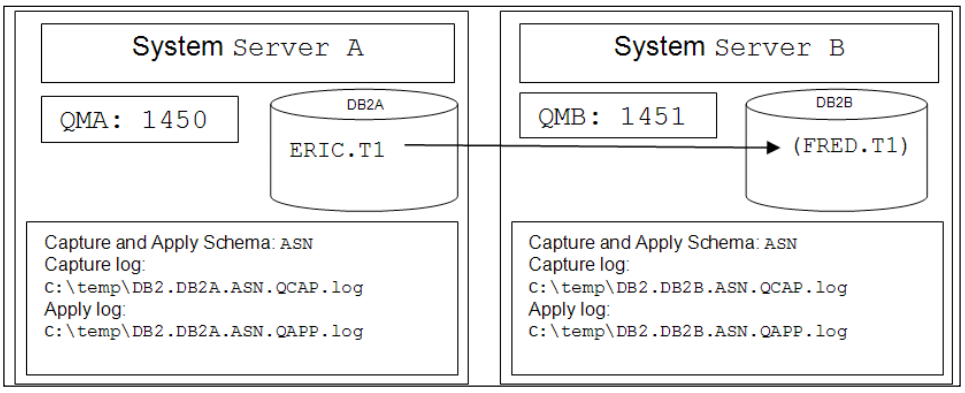
For bidirectional replication:



So now let's move on to look at the various scenarios.

Unidirectional replication

The following diagram shows the setup that we will use:



We will start with a test table `ERIC.T1` which only exists on `DB2A`—Q Apply will create its equivalent called `FRED.T1` on `DB2B`.

The database layer

We need to create one source database `DB2A` and one target database `DB2B`. Follow the instructions in the *First steps – Database creation* section.

We have now completed the database layer and can proceed to the WebSphere MQ layer.

The WebSphere MQ layer

This section deals with the WebSphere MQ layer, which covers creating the Queue Managers and the appropriate queues, and then starting the Listeners and Channels.

Creating the Queue Managers and Queues

We need to create and start two Queue Managers called `QMA` and `QMB`. Follow the instructions in the *First steps – Queue Manager processing* section.

The queues we need on `QMA` for unidirectional replication are in `SYSA_QMA_MQDEFS_UNI_AB.TXT` file, and are shown next:

```
DELETE QLOCAL(CAPA.ADMINQ)
PURGE

DEFINE QLOCAL(CAPA.ADMINQ) +

REPLACE +

DESCR('LOCAL DEFN OF ADMINQ
FOR CAPA CAPTURE') +

PUT(ENABLED) +

GET(ENABLED) +

SHARE +

DEFSOPT(SHARED) +

DEFPSIST(YES)


DELETE QLOCAL(CAPA.RESTARTQ)
PURGE

DEFINE QLOCAL(CAPA.RESTARTQ)
+

REPLACE +
```

```
DEFINE QREMOTE(CAPA.TO.APPB.
SENDQ.REMOTE) +

REPLACE +

DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPA TO APPB') +

PUT(ENABLED) +

XMITQ(QMB.XMITQ)  +

RNAME(CAPA.TO.APPB.RECVQ) +

RQMNAME(QMB) +

DEFPSIST(YES)


DEFINE QLOCAL(QMB.XMITQ) +

REPLACE +

DESCR('TRANSMISSION QUEUE TO
QMB') +

USAGE(XMITQ) +
```

```
DESCR('LOCAL DEFN OF RESTART FOR
CAPA CAPTURE') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)

DEFINE QLOCAL(DEAD.LETTER.QUEUE.
QMA) +

REPLACE +
DESCR('LOCAL DEAD LETTER QUEUE
QMA') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)
```

```
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
TRIGTYPE(FIRST) +
TRIGDATA(QMA.TO.QMB) +
INITQ(SYSTEM.CHANNEL.INITQ)

DEFINE CHANNEL(QMA.TO.QMB) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMB') +
XMITQ(QMB.XMITQ) +
CONNAME('127.0.0.1(1451)')

DEFINE CHANNEL(QMB.TO.QMA) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL
FROM QMB')
```

From CLP-B, run the file as:

```
$ runmqsc QMB < SYSB_QMB_MQDEFS_UNI_BA.TXT
```

The queues we need on QMB for unidirectional replication are in SYSB_QMB_MQDEFS_UNI_BA.TXT file, and are shown next:

```
DEFINE QMODEL(IBMQREP.SPILL.
MODELQ) +
REPLACE +
SHARE +
DEFSOPT(SHARED) +
MAXDEPTH(500000) +
MAXMSGL(500000) +
MSGDLVSQ(FIFO) +
DEFTYPE(PERMDYN)

DEFINE QLOCAL(DEAD.LETTER.
QUEUE.QMB) +
REPLACE +
DESCR('LOCAL DEAD LETTER QUEUE
QMB') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)

DEFINE QLOCAL(CAPA.TO.APPB.
RECVQ) +
REPLACE +
DESCR('LOCAL RECEIVE QUEUE -
APPB FROM CAPA') +
PUT(ENABLED) +
GET(ENABLED) +
DEFINE QLOCAL(QMA.XMITQ) +
REPLACE +
DESCR('TRANSMISSION QUEUE TO
QMA') +
USAGE(XMITQ) +
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
TRIGTYPE(FIRST) +
TRIGDATA(QMB.TO.QMA) +
INITQ(SYSTEM.CHANNEL.INITQ)

DEFINE CHANNEL(QMB.TO.QMA) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMA') +
XMITQ(QMA.XMITQ) +
CONNAME('127.0.0.1(1450)')

DEFINE CHANNEL(QMA.TO.QMB) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM
QMA')
```

```
DEFSOPT(SHARED) +

DEFPSIST(YES)


DEFINE QREMOTE(CAPA.ADMINQ.
REMOTE) +

REPLACE +

DESCR('REMOTE DEFN OF ADMINQ FOR
CAPA CAPTURE') +

PUT(ENABLED) +

XMITQ(QMA.XMITQ) +

RNAME(CAPA.ADMINQ) +

RQMNAME(QMA) +

DEFPSIST(YES)
```

From CLP-B, run the file as:

```
$ runmqsc QMB < SYSB_QMB_MQDEFS_UNI_BA.TXT
```

Starting the Listeners

To start the Listeners for `QMA` and `QMB`, follow the instructions in the *MQ Listener management – Defining/Starting an MQ Listener* section of *Chapter 4*.

Starting the Channels

To start the Channels for `QMA` and `QMB`, follow the instructions in the *MQ Channel management – To start a Channel* section of *Chapter 4*.

Testing the WebSphere MQ layer

Now that everything is started, we need to test the MQ layer.

We will start by putting test messages onto each system using the `amqsput` command and then retrieving them using the `amqsget` command.

The put message file is called `SYSA_QMA_TEST1.TXT` and its contents are described in *First steps – Text file creation for the amqsput command* section. The batch file is called `SYSA_QMA_TESTP_UNI_AB.BAT` and contains:

```
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput" CAPA.TO.APPB.
SENDQ.REMOTE     QMA  < SYSA_QMA_TEST1.TXT
```

From CLP-A, run the file as:

```
$ SYSA_QMA_TESTP_UNI_AB.BAT
```

The put message file for `QMB` is called `SYSB_QMB_TEST2.TXT` and the batch file is called `SYSB_QMB_TESTP_UNI_BA.BAT` and contains:

```
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput" CAPA.ADMINQ.
REMOTE     QMB  < SYSB_QMB_TEST2.TXT
```

From CLP-B, run the file as:

```
$ SYSB_QMB_TESTP_UNI_BA.BAT
```

Once we have put the test messages onto each system, we can retrieve them.

The get message batch file for `QMA` is called `SYSA_QMA_TESTG_UNI_BA.BAT` and contains:

```
@ECHO This takes 15 seconds to run
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget"  CAPA.ADMINQ  QMA
@ECHO You should see: test2
```

From CLP-A, run the file as:

```
$ SYSA_QMA_TESTG_UNI_BA.BAT
```

The get message batch file for `QMB` is called `SYSB_QMB_TESTG_UNI_AB.BAT` and contains:

```
@ECHO This takes 15 seconds to run
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget"  CAPA.TO.APPB.
RECVQ   QMB
@ECHO You should see: test1
```

From CLP-B, run the file as:

```
$ SYSB_QMB_TESTG_UNI_AB.BAT
```

Provided we see the messages that we are told we should see, then we have successfully tested the WebSphere MQ layer.

We have now defined the database and WebSphere MQ layers, and can proceed to the Q replication layer.

The Q replication layer

The following sections cover the steps to create the control tables, the Replication Queue Maps, and the Q subscription. The tasks are:

- Creating the Q Capture tables on `DB2A`
- Creating the Q Apply tables on `DB2B`
- Creating a Replication Queue Map for `DB2A` to `DB2B`
- Creating a Q subscription

Creating Q Capture control tables on DB2A

Follow the instructions in the *Common Q replication tasks – Creating or dropping Q Capture control tables on DB2A* section of *Chapter 5*, *The ASNCLP Command Interface*.

Creating Q Apply control tables on DB2B

Follow the instructions in the *Common Q replication tasks – Creating or dropping Q Apply control tables on DB2B* section of *Chapter 5*.

Creating a Replication Queue Map from DB2A to DB2B

Follow the instructions in the *Queue Map maintenance—Creating a Replication Queue Map* section of *Chapter 5*.

Creating a Q subscription

Follow the instructions in the *Creating Q subscriptions and Publications – Q subscription for unidirectional replication* section of *Chapter 5*.

Starting Q Capture and Q Apply

Now we need to start Q Capture and Q Apply.

Starting Q Capture on DB2A

To start Q Capture, follow the instructions in the *Q Capture administration – Starting Q Capture* section of *Chapter 6*.

Starting Q Apply on DB2B

To start Q Apply, follow the instructions in the *Q Apply administration – Starting Q Apply* section of *Chapter 6*.

Testing replication

We are now in a position to test the unidirectional replication setup. We can insert a record into `ERIC.T1` on `DB2A` and check that it is replicated to `FRED.T1` on `DB2B`.

From CLP-A, issue:

```
$ db2 "insert into eric.t1 values (1,1,'H')"
```

From CLP-B, issue:

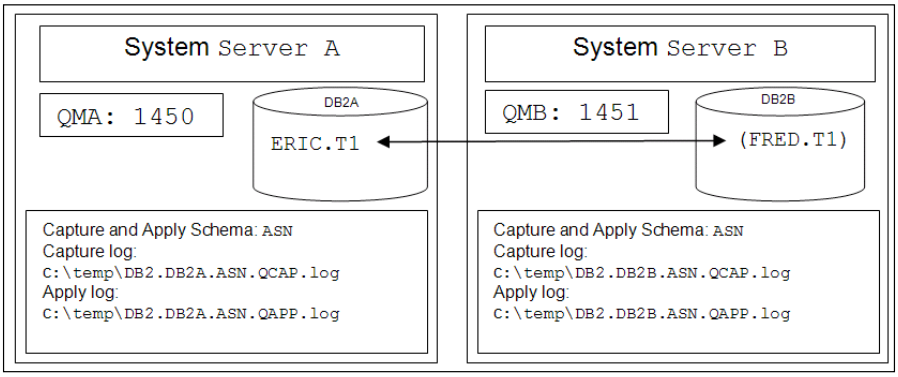
```
$ db2 "select * from fred.t1"

C1          C2          C3
----------- ----------- ----------
          1           1 H
  1 record(s) selected.
```

We should see one record in `fred.t1` on `DB2B`.

We can see that the unidirectional Q replication setup is working.

Bidirectional replication

The following diagram shows the setup that we will use:



We will start with a test table `ERIC.T1` which only exists on `DB2A`—Q Apply will create its equivalent called `FRED.T1` on `DB2B`.

The database layer

We need to create two source databases called `DB2A` and `DB2B`—follow the instructions in the *First steps—Database creation* section.

We now have the database layer defined and can proceed to the WebSphere MQ layer.

The WebSphere MQ layer

This section deals with the WebSphere MQ layer, which covers creating the Queue Managers and the appropriate queues, and then starting the Listeners and Channels.

Creating the Queue Managers and Queues

We need to create and start two Queue Managers called `QMA` and `QMB`—follow the instructions in the *First steps – Queue Manager processing* section.

We now need to create the queues that we need for this type of replication. Note that the queues we need for bidirectional replication are the same as for P2P two-way replication.

The queues we need for `QMB` are in `SYSB_QMB_MQDEFS_BIP2P2W_AB.TXT` file, and are shown next:

```
DELETE QLOCAL(CAPA.ADMINQ) PURGE        *

DEFINE QLOCAL(CAPA.ADMINQ) +            DEFINE QLOCAL(CAPB.TO.APPA.
REPLACE +                               RECVQ) +
DESCR('LOCAL DEFN OF ADMINQ FOR CAPA    REPLACE +
CAPTURE') +                             DESCR('LOCAL RECEIVE QUEUE -
PUT(ENABLED) +                          APPA FROM CAPB') +
GET(ENABLED) +                          PUT(ENABLED) +
SHARE +                                 GET(ENABLED) +
DEFSOPT(SHARED) +                       DEFSOPT(SHARED) +
DEFPSIST(YES)                           DEFPSIST(YES)
```

```
*

DELETE QLOCAL(CAPA.RESTARTQ)
PURGE

DEFINE QLOCAL(CAPA.RESTARTQ) +

REPLACE +

DESCR('LOCAL DEFN OF RESTART FOR
CAPA CAPTURE') +

PUT(ENABLED) +

GET(ENABLED) +

SHARE +

DEFSOPT(SHARED) +

DEFPSIST(YES)

*

DEFINE QMODEL(IBMQREP.SPILL.
MODELQ) +

REPLACE +

SHARE +

DEFSOPT(SHARED) +

MAXDEPTH(500000) +

MAXMSGL(500000) +

MSGDLVSQ(FIFO) +

DEFTYPE(PERMDYN)

*

DEFINE QLOCAL(DEAD.LETTER.QUEUE.
QMA)

+

REPLACE +

DESCR('LOCAL DEAD LETTER QUEUE
QMA') +
```

```
*

DEFINE QREMOTE(CAPB.ADMINQ.
REMOTE) +

REPLACE +

DESCR('REMOTE DEFN OF ADMINQ FOR
CAPB CAPTURE') +

PUT(ENABLED) +

XMITQ(QMB.XMITQ) +

RNAME(CAPB.ADMINQ) +

RQMNAME(QMB) +

DEFPSIST(YES)

*

DEFINE QLOCAL(QMB.XMITQ) +

REPLACE +

DESCR('TRANSMISSION QUEUE TO
QMB') +

USAGE(XMITQ) +

PUT(ENABLED) +

GET(ENABLED) +

TRIGGER +

TRIGTYPE(FIRST) +

TRIGDATA(QMA.TO.QMB) +

INITQ(SYSTEM.CHANNEL.INITQ)

*

DEFINE CHANNEL(QMA.TO.QMB) +

CHLTYPE(SDR) +

REPLACE +
```

```
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QREMOTE(CAPA.TO.APPB.
SENDQ.REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPA TO APPB') +
PUT(ENABLED) +
XMITQ(QMB.XMITQ)  +
RNAME(CAPA.TO.APPB.RECVQ) +
RQMNAME(QMB) +
DEFPSIST(YES)
```

```
TRPTYPE(TCP) +
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMB')
+
XMITQ(QMB.XMITQ) +
CONNAME('127.0.0.1(1451)')
*
DEFINE CHANNEL(QMB.TO.QMA) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM
QMB')
*
```

From CLP-A, issue:

```
$ runmqsc QMA < SYSA_QMA_MQDEFS_BIP2P2W_AB.TXT
```

The queues we need for `QMB` are in `SYSB_QMB_MQDEFS_BIP2P2W_AB.TXT` file, and are shown next:

```
DELETE QLOCAL(CAPB.ADMINQ) PURGE
DEFINE QLOCAL(CAPB.ADMINQ) +
REPLACE +
DESCR('LOCAL DEFN OF ADMINQ FOR CAPB
CAPTURE') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
```

```
DEFINE QLOCAL(CAPA.TO.APPB.
RECVQ) +
REPLACE +
DESCR('LOCAL RECEIVE QUEUE
- APPB FROM CAPA') +
PUT(ENABLED) +
GET(ENABLED) +
DEFSOPT(SHARED) +
DEFPSIST(YES)
```

```
DEFPSIST(YES)

DELETE QLOCAL(CAPB.RESTARTQ)
PURGE
DEFINE QLOCAL(CAPB.RESTARTQ) +
REPLACE +
DESCR('LOCAL DEFN OF RESTART FOR
CAPB CAPTURE') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)

DEFINE QMODEL(IBMQREP.SPILL.
MODELQ) +
REPLACE +
SHARE +
DEFSOPT(SHARED) +
MAXDEPTH(500000) +
MAXMSGL(500000) +
MSGDLVSQ(FIFO) +
DEFTYPE(PERMDYN)

DEFINE QLOCAL(DEAD.LETTER.QUEUE.
QMB) +
REPLACE +
DESCR('LOCAL DEAD LETTER QUEUE
QMB') +
```

```
DEFINE QREMOTE(CAPA.ADMINQ.
REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF ADMINQ
FOR CAPA CAPTURE') +
PUT(ENABLED) +
XMITQ(QMA.XMITQ) +
RNAME(CAPA.ADMINQ) +
RQMNAME(QMA) +
DEFPSIST(YES)

DEFINE QLOCAL(QMA.XMITQ) +
REPLACE +
DESCR('TRANSMISSION QUEUE TO
QMA') +
USAGE(XMITQ) +
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
TRIGTYPE(FIRST) +
TRIGDATA(QMB.TO.QMA) +
INITQ(SYSTEM.CHANNEL.INITQ)

DEFINE CHANNEL(QMB.TO.QMA) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
```

```
PUT(ENABLED) +
GET(ENABLED) +
SHARE +

DEFSOPT(SHARED) +
DEFPSIST(YES)

DEFINE QREMOTE(CAPB.TO.APPA.
SENDQ.REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPB TO APPA') +
PUT(ENABLED) +
XMITQ(QMA.XMITQ)  +
RNAME(CAPB.TO.APPA.RECVQ) +
RQMNAME(QMA) +
DEFPSIST(YES)
```

```
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMA') +
XMITQ(QMA.XMITQ) +
CONNAME('127.0.0.1(1450)')

DEFINE CHANNEL(QMA.TO.QMB) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM
QMA')
```

From CLP-B, run the file as:

```
$ runmqsc QMB < SYSB_QMB_MQDEFS_BIP2P2W_AB.TXT
```

Starting the Listeners

To start the Listeners, follow the instructions in the *The WebSphere MQ layer – Start the Listeners* section.

Starting the Channels

To start the Channels, follow the instructions in the *The WebSphere MQ layer – Start the Channels* section.

Testing the WebSphere MQ layer

Now that everything is started, we need to test the MQ layer.

We will start by putting test messages onto each system using the `amqsput` command and then retrieving them using the `amqsget` command.

The put message batch file for `QMA` is called `SYSA_QMA_TESTP_BIP2P2W.BAT` and contains:

```
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPA.TO.APPB.
SENDQ.REMOTE     QMA < SYSA_QMA_TEST1.TXT
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPB.ADMINQ.
REMOTE     QMA < SYSA_QMA_TEST2.TXT
```

From CLP-A, run the file as:

```
$ SYSA_QMA_TESTP_BIP2P2W.BAT
```

The put message file for `QMB` is called `SYSB_QMB_TESTP_BIP2P2W.BAT` and contains:

```
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPB.TO.APPA.
SENDQ.REMOTE     QMB < SYSB_QMB_TEST3.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPA.ADMINQ.
REMOTE     QMB < SYSB_QMB_TEST4.TXT
```

From CLP-B, run the file as:

```
$ SYSB_QMB_TESTP_BIP2P2W.BAT
```

Once we have put the test messages onto each system, we can retrieve them.

The get message file for `QMA` is called `SYSA_QMA_TESTG_BIP2PW.BAT` and contains:

```
@echo The amqsget program take 15 seconds to run
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPA.ADMINQ QMA

@ECHO You should see above:  test4

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPB.TO.APPA.
RECVQ QMA

@ECHO You should see above:  test3
```

From CLP-A, run the file as:

```
$ SYSA_QMA_TESTG_BIP2P2W.BAT
```

The get message file for QMB is called SYSB_QMB_TESTG_BIP2PW.BAT and contains:

```
@echo The amqsget program take 15 seconds to run
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPA.TO.APPB.
RECVQ QMB
@ECHO You should see above:  test1
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPB.ADMINQ QMB
@ECHO You should see above:  test2
```

From CLP-B, run the file as:

```
$ SYSB_QMB_TESTG_BIP2P2W.BAT
```

Provided we see the messages that we are told we should see, then we have successfully tested the WebSphere MQ layer, and can proceed to the Q replication layer.

The Q replication layer

The following sections give the ASNCLP commands to create the control tables, the Replication Queue Maps and the Q subscription. The tasks are:

- Creating the Q Capture and Q Apply tables on DB2A
- Creating the Q Capture and Q Apply tables on DB2B
- Creating a Replication Queue Map for DB2A to DB2B
- Creating a Replication Queue Map for DB2B to DB2A
- Creating a Q subscription

Creating Q Capture/Q Apply control tables on DB2A

Follow the instructions in the *Common Q replication tasks – Creating control tables in the same database* section of *Chapter 5*, *The ASNCLP Command Interface*.

Creating Q Capture/Q Apply control tables on DB2B

Follow the instructions in the *Common Q replication tasks – Creating control tables in the same database* section of *Chapter 5*.

Creating a Replication Queue Map for DB2A to DB2B

Follow the instructions in the *Queue Map maintenance – Creating a Replication Queue Map* section of *Chapter 5*.

Creating a Replication Queue Map for DB2B to DB2A

Follow the instructions in the *Queue Map maintenance – Creating a Replication Queue Map* section of *Chapter 5*.

Creating a bidirectional Q subscription

Follow the instructions in the *Creating Q subscriptions and Publications – Q subscription for bidirectional replication* section of *Chapter 5*.

Starting Q Capture and Q Apply

Now we need to start Q Capture and Q Apply.

Starting Q Capture on DB2A and DB2B

Follow the instructions in the *Q Capture administration – Starting Q Capture* section of *Chapter 6*, *Administration Tasks*.

Wait for both Q Captures to be up and running before starting the Q Applys.

Starting Q Apply on DB2A and DB2B

Follow the instructions in the *Q Apply administration – Starting Q Apply* section of *Chapter 6*.

Testing replication

We are now in a position to test the bidirectional replication setup. We can insert a record into `ERIC.T1` on `DB2A` and check that it is replicated to `FRED.T1` on `DB2B`.

From CLP-A, issue:

```
$ db2 "insert into eric.t1 values (1,1,'H')"
```

From CLP-B, issue:

```
$ db2 "select * from fred.t1"
```

```
C1          C2          C3
----------- ----------- ----------
          1           1 H
  1 record(s) selected.
```

We should see one record in `FRED.T1` on `DB2B`.

And we can insert a record into `FRED.T1` on `DB2B` and check that it is replicated to `ERIC.T1` on `DB2A`.

From CLP-B, issue:

```
$ db2 "insert into fred.t1 values (2,1,'J')"
```

From CLP-A, issue:

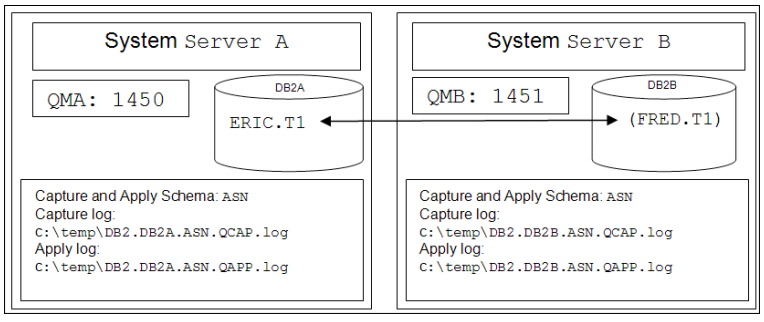
```
$ db2 "select * from eric.t1"

C1          C2          C3
----------- ----------- ----------
          1           1 H
          2           1 J
  2 record(s) selected.
```

We should see two records in `ERIC.T1` on `DB2A`.

We can see that the bidirectional Q replication setup is working.

P2P two-way replication

The setup for P2P two-way is nearly identical to the setup for bidirectional replication. The only difference between them is the way the Q subscription is defined—for bidirectional, we specify `subtype b` and for P2P, we specify `subtype p`. The setup we will use is shown in the following diagram:



We will start with a test table `ERIC.T1` which only exists on `DB2A`—Q Apply will create its equivalent called `FRED.T1` on `DB2B`.

The database layer

The database layer we need for P2P two-way replication is the same as for bidirectional replication in the previous the *Bidirectional replication – The database layer* section.

The WebSphere MQ layer

The queues we need for P2P two-way replication are the same as for bidirectional replication in the previous the *Bidirectional replication – The WebSphere MQ layer* section.

The Q replication layer

The following sections give the ASNCLP commands to create the control tables, the Replication Queue Maps and the Q subscription. The tasks are:

- Creating the Q Capture and Q Apply tables on `DB2A`
- Creating the Q Capture and Q Apply tables on `DB2B`
- Creating a Replication Queue Map for `DB2A` to `DB2B`
- Creating a Replication Queue Map for `DB2B` to `DB2A`
- Creating a Q subscription

Creating Q Capture/Q Apply control tables on DB2A

Follow the instructions in the *Bidirectional replication – The Q replication layer – Creating Q Capture/Q Apply control tables on DB2A* section.

Creating Q Capture/Q Apply control tables on DB2B

Follow the instructions in the *Bidirectional replication – The Q replication layer – Creating Q Capture/Q Apply control tables on DB2B* section.

Creating a Replication Queue Map for DB2A to DB2B

Follow the instructions in the *Bidirectional replication – The Q replication layer – Creating a Replication Queue Map for DB2A to DB2B* section.

Creating a Replication Queue Map for DB2B to DB2A

Follow the instructions in the *Bidirectional replication – The Q replication layer – Creating a Replication Queue Map for DB2B to DB2A* section.

Creating a Q subscription

To create a P2P two-way Q subscription, follow the instructions in the *Creating Q subscriptions and Publications – Q subscription for P2P two-way replication* section of *Chapter 5*.

Starting Q Capture and Q Apply

Now we need to start Q Capture and Q Apply.

Starting Q Capture on DB2A and DB2B

To start Q Capture, follow the instructions in the *Q Capture administration—Starting Q Capture* section of *Chapter 6*.

Wait for both Q Captures to be up and running before starting the Q Applys.

Starting Q Apply on DB2A and DB2B

To start Q Apply, follow the instructions in *Q Apply administration – Starting Q Apply* section of *Chapter 6*.

Testing replication

We are now in a position to test our P2P two-way replication setup. We can insert a record into `ERIC.T1` on `DB2A` and check that it is replicated to `FRED.T1` on `DB2B`.

From CLP-A, issue:

```
$ db2 "insert into eric.t1(c1,c2,c3) values (1,1,'H')"
```

From CLP-B, issue:

```
$ db2 "select * from fred.t1"

C1          C2          C3          ibmqrepVERTIME
ibmqrepVERNODE
----------- ----------- ----------- ----------------------------------
          1           1 H           2007-03-21-13.45.25.640000       4
```

We should see one record in `FRED.T1` on `DB2B`.

Note the two extra columns which have automatically been added to the source and target tables.

And then we insert a record into `FRED.T1` on `DB2B` and check that it is replicated to `ERIC.T1` on `DB2A`.

From CLP-B, issue:

```
$ db2 "insert into fred.t1(c1,c2,c3) values (2,1,'J')"
```

From CLP-A, issue:

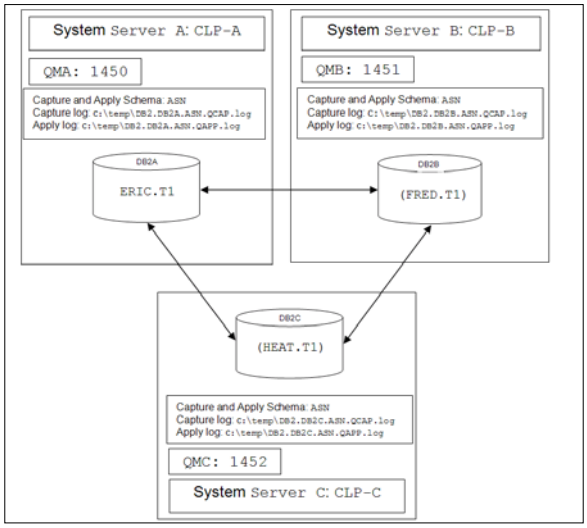
```
$ db2 "select * from eric.t1"

C1          C2          C3          ibmqrepVERTIME
ibmqrepVERNODE
----------- ----------- ---------- -------------------------- -------
          1           1 H          2007-03-21-13.45.25.640000           4
          2           2 T          2007-03-21-13.45.52.906000           8
```

We should see two records in `ERIC.T1` on `DB2A`.

We can see that the P2P two-way Q replication setup is working.

P2P three-way replication

The setup we will use is shown in the following diagram:



We will start with a test table `ERIC.T1` which only exists on `DB2A`—Q Apply will create the appropriate target tables on `DB2B` and `DB2C`.

The database layer

We need to create three source databases called `DB2A`, `DB2B`, and `DB2C`—follow the instructions in the *First steps – Database creation* section.

We now have the database layer defined and can proceed to the WebSphere MQ layer.

The WebSphere MQ layer

This section deals with the WebSphere MQ layer, which covers creating the Queue Managers and the appropriate queues, and then starting the Listeners and Channels.

Creating the Queue Managers and Queues

We need to create and start three Queue Managers called `QMA`, `QMB`, and `QMC`—follow the instructions in the *First steps – Queue Manager processing* section.

The queues we need for P2P three-way replication are shown next.

The queues we need for `QMA` are in `SYSA_QMA_MQDEFS_P2P3W_ABC.TXT` f file, and contains:

```
DELETE QLOCAL(CAPA.ADMINQ) PURGE
DEFINE QLOCAL(CAPA.ADMINQ) +
REPLACE +
DESCR('LOCAL DEFN OF ADMINQ FOR
CAPA CAPTURE') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE CHANNEL(QMA.TO.QMB) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMB') +
XMITQ(QMB.XMITQ) +
CONNAME('127.0.0.1(1451)')
*
DEFINE CHANNEL(QMB.TO.QMA) +
CHLTYPE(RCVR) +
```

```
DELETE QLOCAL(CAPA.RESTARTQ)
PURGE

DEFINE QLOCAL(CAPA.RESTARTQ) +

REPLACE +

DESCR('LOCAL DEFN OF RESTART FOR
CAPA CAPTURE') +

PUT(ENABLED) +

GET(ENABLED) +

SHARE +

DEFSOPT(SHARED) +

DEFPSIST(YES)

*

DEFINE QMODEL(IBMQREP.SPILL.
MODELQ) +

REPLACE +

SHARE +

DEFSOPT(SHARED) +

MAXDEPTH(500000) +

MAXMSGL(500000) +

MSGDLVSQ(FIFO) +

DEFTYPE(PERMDYN)

*

DEFINE QLOCAL(DEAD.LETTER.QUEUE.
QMA) +

REPLACE +

DESCR('LOCAL DEAD LETTER QUEUE

QMA') +

PUT(ENABLED) +
```

```
REPLACE +

TRPTYPE(TCP) +

DESCR('RECEIVER CHANNEL FROM
QMB')

*

DEFINE QREMOTE(CAPA.TO.APPC.
SENDQ.REMOTE) +

REPLACE +

DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPA TO APPC') +

PUT(ENABLED) +

XMITQ(QMC.XMITQ)  +

RNAME(CAPA.TO.APPC.RECVQ) +

RQMNAME(QMC) +

DEFPSIST(YES)

*

DEFINE QLOCAL(CAPC.TO.APPA.
RECVQ) +

REPLACE +

DESCR('LOCAL RECEIVE QUEUE -
APPA FROM CAPC') +

PUT(ENABLED) +

GET(ENABLED) +

DEFSOPT(SHARED) +

DEFPSIST(YES)

*

DEFINE QREMOTE(CAPC.ADMINQ.
REMOTE) +

REPLACE +
```

```
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QREMOTE(CAPA.TO.APPB.
SENDQ.REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPA TO APPB') +
PUT(ENABLED) +
XMITQ(QMB.XMITQ)  +
RNAME(CAPA.TO.APPB.RECVQ) +
RQMNAME(QMB) +
DEFPSIST(YES)
*
DEFINE QLOCAL(CAPB.TO.APPA.RECVQ)
+
REPLACE +
DESCR('LOCAL RECEIVE QUEUE - APPA
FROM CAPB') +
PUT(ENABLED) +
GET(ENABLED) +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QREMOTE(CAPB.ADMINQ.
REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF ADMINQ FOR
CAPC CAPTURE') +
PUT(ENABLED) +
XMITQ(QMC.XMITQ) +
RNAME(CAPC.ADMINQ) +
RQMNAME(QMC) +
DEFPSIST(YES)
*
DEFINE QLOCAL(QMC.XMITQ) +
REPLACE +
DESCR('TRANSMISSION QUEUE TO
QMC') +
USAGE(XMITQ) +
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
TRIGTYPE(FIRST) +
TRIGDATA(QMA.TO.QMC) +
INITQ(SYSTEM.CHANNEL.INITQ)
*
DEFINE CHANNEL(QMA.TO.QMC) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMC') +
XMITQ(QMC.XMITQ) +
CONNAME('127.0.0.1(1452)')
```

```
DESCR('REMOTE DEFN OF ADMINQ FOR
CAPB CAPTURE') +
PUT(ENABLED) +
XMITQ(QMB.XMITQ) +
RNAME(CAPB.ADMINQ) +
RQMNAME(QMB) +
DEFPSIST(YES)
*
DEFINE QLOCAL(QMB.XMITQ) +
REPLACE +
DESCR('TRANSMISSION QUEUE TO
QMB') +
USAGE(XMITQ) +
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
TRIGTYPE(FIRST) +
TRIGDATA(QMA.TO.QMB) +
INITQ(SYSTEM.CHANNEL.INITQ) *
```

```
*
DEFINE CHANNEL(QMC.TO.QMA) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM
QMC')
*
```

From CLP-A, run the file as:

```
$ runmqsc QMA < SYSA_QMA_MQDEFS_P2P3W_ABC.TXT
```

The queues we need for `QMB` are in `SYSB_QMB_MQDEFS_P2P3W_ABC.TXT` file, and contains:

```
DELETE QLOCAL(CAPB.ADMINQ) PURGE
DEFINE QLOCAL(CAPB.ADMINQ) +
REPLACE +
DESCR('LOCAL DEFN OF ADMINQ FOR
CAPB CAPTURE') +
```

```
*
DEFINE CHANNEL(QMB.TO.QMA) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
```

```
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DELETE QLOCAL(CAPB.RESTARTQ)
PURGE
DEFINE QLOCAL(CAPB.RESTARTQ) +
REPLACE +
DESCR('LOCAL DEFN OF RESTART FOR
CAPB CAPTURE') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QMODEL(IBMQREP.SPILL.
MODELQ) +
REPLACE +
SHARE +
DEFSOPT(SHARED) +
MAXDEPTH(500000) +
MAXMSGL(500000) +
MSGDLVSQ(FIFO) +
DEFTYPE(PERMDYN)
*
DEFINE QLOCAL(DEAD.LETTER.QUEUE.
QMB) +
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMA') +
XMITQ(QMA.XMITQ) +
CONNAME('127.0.0.1(1450)')
*
DEFINE CHANNEL(QMA.TO.QMB) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM
QMA')
*
DEFINE QREMOTE(CAPB.TO.APPC.
SENDQ.REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPB TO APPC') +
PUT(ENABLED) +
XMITQ(QMC.XMITQ)  +
RNAME(CAPB.TO.APPC.RECVQ) +
RQMNAME(QMC) +
DEFPSIST(YES)
*
DEFINE QLOCAL(CAPC.TO.APPB.
RECVQ) +
REPLACE +
DESCR('LOCAL RECEIVE QUEUE -
APPB FROM CAPC') +
PUT(ENABLED) +
GET(ENABLED) +
```

```
REPLACE +
DESCR('LOCAL DEAD LETTER QUEUE
QMB') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QREMOTE(CAPB.TO.APPA.
SENDQ.REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPB TO APPA') +
PUT(ENABLED) +
XMITQ(QMA.XMITQ)  +
RNAME(CAPB.TO.APPA.RECVQ) +
RQMNAME(QMA) +
DEFPSIST(YES)
*
DEFINE QLOCAL(CAPA.TO.APPB.RECVQ)
+
REPLACE +
DESCR('LOCAL RECEIVE QUEUE - APPB
FROM CAPA') +
PUT(ENABLED) +
GET(ENABLED) +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QREMOTE(CAPC.ADMINQ.
REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF ADMINQ FOR
CAPC CAPTURE') +
PUT(ENABLED) +
XMITQ(QMC.XMITQ) +
RNAME(CAPC.ADMINQ) +
RQMNAME(QMC) +
DEFPSIST(YES)
*
DEFINE QLOCAL(QMC.XMITQ) +
REPLACE +
DESCR('TRANSMISSION QUEUE TO
QMC') +
USAGE(XMITQ) +
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
TRIGTYPE(FIRST) +
TRIGDATA(QMB.TO.QMC) +
INITQ(SYSTEM.CHANNEL.INITQ)
*
DEFINE CHANNEL(QMB.TO.QMC) +
CHLTYPE(SDR) +
```

```
DEFPSIST(YES)

*

DEFINE QREMOTE(CAPA.ADMINQ.
REMOTE) +

REPLACE +

DESCR('REMOTE DEFN OF ADMINQ FOR
CAPA CAPTURE') +

PUT(ENABLED) +

XMITQ(QMA.XMITQ) +

RNAME(CAPA.ADMINQ) +

RQMNAME(QMA) +

DEFPSIST(YES)

*

DEFINE QLOCAL(QMA.XMITQ) +

REPLACE +

DESCR('TRANSMISSION QUEUE TO
QMA') +

USAGE(XMITQ) +

PUT(ENABLED) +

GET(ENABLED) +

TRIGGER +

TRIGTYPE(FIRST) +

TRIGDATA(QMB.TO.QMA) +

INITQ(SYSTEM.CHANNEL.INITQ)
REPLACE +

TRPTYPE(TCP) +

DISCINT(0) +

DESCR('SENDER CHANNEL TO QMC') +

XMITQ(QMC.XMITQ) +

CONNAME('127.0.0.1(1452)')

*

DEFINE CHANNEL(QMC.TO.QMB) +

CHLTYPE(RCVR) +

REPLACE +

TRPTYPE(TCP) +

DESCR('RECEIVER CHANNEL FROM
QMC')*
```

From CLP-B, run the file as:

```
$ runmqsc QMB < SYSB_QMB_MQDEFS_P2P3W_ABC.TXT
```

The queues we need for QMC are in SYSC_QMC_MQDEFS_P2P3W_ABC.TXT file, and contains:

```
DELETE QLOCAL(CAPC.ADMINQ) PURGE
DEFINE QLOCAL(CAPC.ADMINQ) +
REPLACE +
DESCR('LOCAL DEFN OF ADMINQ FOR CAPC CAPTURE') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DELETE QLOCAL(CAPC.RESTARTQ) PURGE
DEFINE QLOCAL(CAPC.RESTARTQ) +
REPLACE +
DESCR('LOCAL DEFN OF RESTART FOR CAPC CAPTURE') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QMODEL(IBMQREP.SPILL.MODELQ) +
REPLACE +
SHARE +
*
DEFINE CHANNEL(QMC.TO.QMA) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMA') +
XMITQ(QMA.XMITQ) +
CONNAME('127.0.0.1(1450)')
*
DEFINE CHANNEL(QMA.TO.QMC) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM QMA')
*
DEFINE QREMOTE(CAPC.TO.APPB.SENDQ.REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF SEND QUEUE FROM CAPC TO APPB') +
PUT(ENABLED) +
XMITQ(QMB.XMITQ)  +
RNAME(CAPC.TO.APPB.RECVQ) +
RQMNAME(QMB) +
DEFPSIST(YES)
```

```
DEFSOPT(SHARED) +
MAXDEPTH(500000) +
MAXMSGL(500000) +
MSGDLVSQ(FIFO) +
DEFTYPE(PERMDYN)
*
DEFINE QLOCAL(DEAD.LETTER.QUEUE.
QMC) +
REPLACE +
DESCR('LOCAL DEAD LETTER QUEUE
QMC') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QREMOTE(CAPC.TO.APPA.
SENDQ.REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPC TO APPA') +
PUT(ENABLED) +
XMITQ(QMA.XMITQ)  +
RNAME(CAPC.TO.APPA.RECVQ) +
RQMNAME(QMA) +
DEFPSIST(YES)
*
DEFINE QLOCAL(CAPA.TO.APPC.RECVQ)
*
DEFINE QLOCAL(CAPB.TO.APPC.
RECVQ) +
REPLACE +
DESCR('LOCAL RECEIVE QUEUE -
APPC FROM CAPB') +
PUT(ENABLED) +
GET(ENABLED) +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QREMOTE(CAPB.ADMINQ.
REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF ADMINQ FOR
CAPB CAPTURE') +
PUT(ENABLED) +
XMITQ(QMB.XMITQ) +
RNAME(CAPB.ADMINQ) +
RQMNAME(QMB) +
DEFPSIST(YES)
*
DEFINE QLOCAL(QMB.XMITQ) +
REPLACE +
DESCR('TRANSMISSION QUEUE TO
QMB') +
USAGE(XMITQ) +
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
```

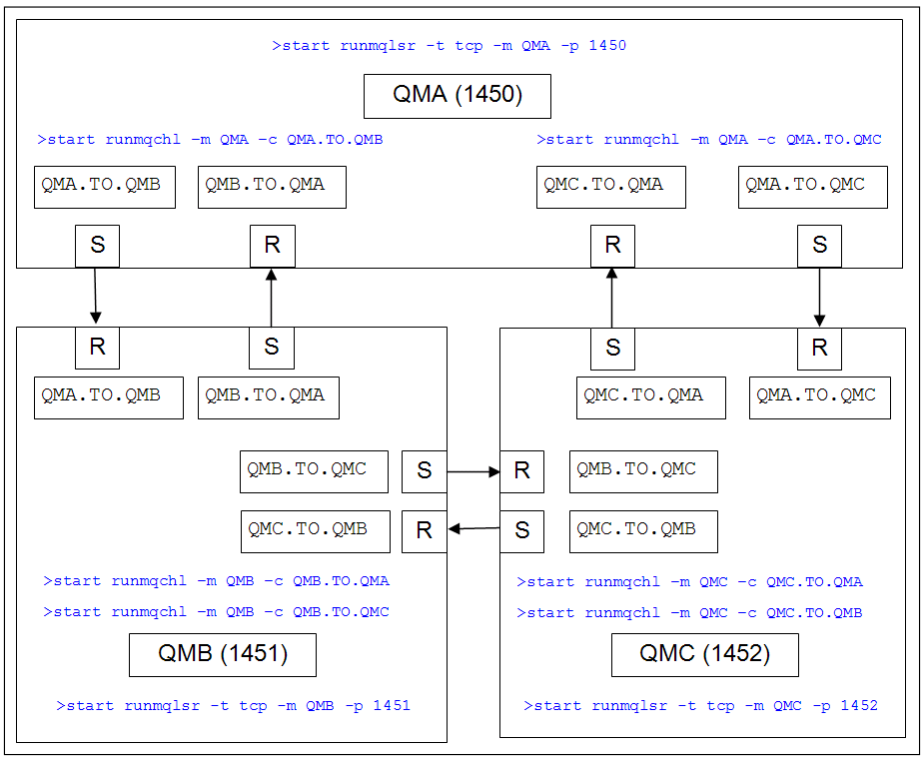
```
+
REPLACE +
DESCR('LOCAL RECEIVE QUEUE - APPC
FROM CAPA') +
PUT(ENABLED) +
GET(ENABLED) +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QREMOTE(CAPA.ADMINQ.
REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF ADMINQ FOR
CAPA CAPTURE') +
PUT(ENABLED) +
XMITQ(QMA.XMITQ) +
RNAME(CAPA.ADMINQ) +
RQMNAME(QMA) +
DEFPSIST(YES)
*
DEFINE QLOCAL(QMA.XMITQ) +
REPLACE +
DESCR('TRANSMISSION QUEUE TO
QMA') +
USAGE(XMITQ) +
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
TRIGTYPE(FIRST) +
TRIGDATA(QMC.TO.QMA) +
INITQ(SYSTEM.CHANNEL.INITQ)
TRIGTYPE(FIRST) +
TRIGDATA(QMC.TO.QMB) +
INITQ(SYSTEM.CHANNEL.INITQ)
*
DEFINE CHANNEL(QMC.TO.QMB) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMB')
+
XMITQ(QMB.XMITQ) +
CONNAME('127.0.0.1(1451)')
*
DEFINE CHANNEL(QMB.TO.QMC) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM
QMB')
*
```

From CLP-C, run the file as:

```
$ runmqsc QMC < SYSC_QMC_MQDEFS_P2P3W_ABC.TXT
```

Starting the Listeners

The Listeners and Channels for P2P three-way replication is shown in the following diagram:



Start the Listeners for `QMA` and `QMB` as described in the *Bidirectional replication – The WebSphere MQ layer – Starting the Listeners* section.

The Listener for `QMC` can be started using the `SYSC_QMC_START_RUNMQLSR.BAT` batch file as shown next:

```
start runmqlsr -t tcp -m  QMC   -p  1452
```

From CLP-C, run the file as:

```
$ SYSC_QMC_START_RUNMQLSR.BAT
```

Starting the Channels

Start the Channels between QMA and QMB as described in the *Bidirectional replication – The WebSphere MQ layer – Starting the Channels* section.

The Channel from QMA to QMC can be started by executing the contents of the batch file called SYSA_QMA_START_RUNMQCHL_AC.BAT:

```
start runmqchl -m  QMA -c QMA.TO.QMC
```

From CLP-A, run the file as:

```
$ SYSA_QMA_START_RUNMQCHL_AC.BAT
```

The Channel from QMB to QMC can be started by executing the contents of the SYSB_QMB_START_RUNMQCHL_BC.BAT batch file:

```
start runmqchl -m  QMB -c QMB.TO.QMC
```

From CLP-B, run the file as:

```
$ SYSB_QMB_START_RUNMQCHL_BC.BAT
```

The Channel from QMC to QMA can be started by executing the contents of the SYSC_QMC_START_RUNMQCHL_CA.BAT batch file:

```
start runmqchl -m  QMC -c QMC.TO.QMA
```

From CLP-C, run the file as:

```
$ SYSC_QMC_START_RUNMQCHL_CA.BAT
```

The Channel from QMC to QMB can be started by executing the contents of the SYSC_QMC_START_RUNMQCHL_CB.BAT file:

```
start runmqchl -m  QMC -c QMC.TO.QMB
```

From CLP-C, run the file as:

```
$ SYSC_QMC_START_RUNMQCHL_CB.BAT
```

Testing the WebSphere MQ layer

Now that everything is started we need to test the MQ layer.

We will start by putting test messages onto each system using the amqsput command and then retrieving them using the amqsget command.

The following three commands will put messages onto the appropriate Send Queues.

1. The put message file for QMA is called SYSA_QMA_TESTP_P2P3W.BAT and contains:

```
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPA.
TO.APPB.SENDQ.REMOTE     QMA < SYSA_QMA_TEST1.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPB.ADMINQ.
REMOTE     QMA <SYSA_QMA_TEST2.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPA.
TO.APPC.SENDQ.REMOTE     QMA < SYSA_QMA_TEST3.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPC.ADMINQ.
REMOTE     QMA < SYSA_QMA_TEST4.TXT
```

   From CLP-A, run the file as:

```
$ SYSA_QMA_TESTP_P2P3W.BAT
```

2. The put message file for QMB is called SYSB_QMB_TESTP_P2P3W.BAT and contains:

```
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPB.
TO.APPA.SENDQ.REMOTE     QMB < SYSB_QMB_TEST5.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPA.ADMINQ.
REMOTE     QMB < SYSB_QMB_TEST6.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPB.
TO.APPC.SENDQ.REMOTE     QMB < SYSB_QMB_TEST7.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPC.ADMINQ.
REMOTE     QMB < SYSB_QMB_TEST8.TXT
```

   From CLP-B, run the file as:

```
$ SYSB_QMB_TESTP_P2P3W.BAT
```

3. The put message file for QMC is called SYSC_QMC_TESTP_P2P3W.BAT and contains:

```
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPC.
TO.APPA.SENDQ.REMOTE     QMC < SYSC_QMC_TEST9.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPA.ADMINQ.
REMOTE     QMC < SYSC_QMC_TEST10.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPC.
TO.APPB.SENDQ.REMOTE     QMC < SYSC_QMC_TEST11.TXT
```

```
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPB.ADMINQ.
REMOTE     QMC < SYSC_QMC_TEST12.TXT
```

From CLP-C, run the file as:

```
$ SYSC_QMC_TESTP_P2P3W.BAT
```

Once we have put the test messages onto each Send Queue, we can retrieve them from the appropriate Receive Queue.

1. The get message file for `QMA` is called `SYSA_QMA_TESTG_P2P3W.BAT` and contains:

```
@echo The amqsget program take 15 seconds to run
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPA.ADMINQ
QMA

@ECHO You should see above:  test6 test10

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPB.TO.APPA.
RECVQ QMA

@ECHO You should see above:  test5

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPC.TO.APPA.
RECVQ QMA

@ECHO You should see above:  test9
```

   From CLP-A, run the file as:

```
$ SYSA_QMA_TESTG_P2P3W.BAT
```

2. The get message file for `QMB` is called `SYSB_QMB_TESTG_P2P3W.BAT` and contains:

```
@echo The amqsget program take 15 seconds to run
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPA.TO.APPB.
RECVQ QMB

@ECHO You should see above:  test1

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPB.ADMINQ
QMB

@ECHO You should see above:  test2 test12

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPC.TO.APPB.
RECVQ QMB

@ECHO You should see above:  test11
```

   From CLP-B, run the file as:

```
$ SYSB_QMB_TESTG_P2P3W.BAT
```

3. The get message file for QMC is called SYSC_QMC_TESTG_P2P3W.BAT and contains:

```
@echo The amqsget program take 15 seconds to run
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPA.TO.APPC.
RECVQ QMC

@ECHO You should see above:  test3

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPB.TO.APPC.
RECVQ QMC

@ECHO You should see above:  test7

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPC.ADMINQ
QMC

@ECHO You should see above:  test4 test8
```

   From CLP-C, run the file as:

```
$ SYSC_QMC_TESTG_P2P3W.BAT
```

Provided we see the messages that we are told we should see, then we have successfully tested the WebSphere MQ layer.

We have now defined the database and WebSphere MQ layers, and can proceed to the Q replication layer.

The Q replication layer

The following sections give the ASNCLP commands to create the control tables, the Replication Queue Maps, and the Q subscription. The tasks are:

- Creating the Q Capture and Q Apply control tables on DB2A
- Creating the Q Capture and Q Apply control tables on DB2B
- Creating the Q Capture and Q Apply control tables on DB2C
- Creating a Replication Queue Map for DB2A to DB2B
- Creating a Replication Queue Map for DB2A to DB2C
- Creating a Replication Queue Map for DB2B to DB2A
- Creating a Replication Queue Map for DB2B to DB2C
- Creating a Replication Queue Map for DB2C to DB2A
- Creating a Replication Queue Map for DB2C to DB2B
- Creating a Q subscription

Creating Q Capture/Q Apply control tables on DB2A

Follow the instructions in the *Bidirectional replication – The Q replication layer – Creating Q Capture/Q Apply control tables on DB2A* section.

Creating Q Capture/Q Apply control tables on DB2B

Follow the instructions in the *Bidirectional replication – The Q replication layer – Creating Q Capture/Q Apply control tables on DB2B* section.

Creating Q Capture/Q Apply control tables on DB2C

Follow the instructions in the *Bidirectional replication – The Q replication layer – Creating Q Capture/Q Apply control tables on DB2B* section.

Creating a Replication Queue Map for DB2A to DB2B

Follow the instructions in the *Bidirectional replication – The Q replication layer – Creating a Replication Queue Map for DB2A to DB2B* section.

Creating a Replication Queue Map for DB2A to DB2C

The Replication Queue Map for the queues going from `DB2A` to `DB2C` can be created using the following ASNCLP commands in the `SYSA_crt_rqma2c.asnclp` file:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2A;
SET SERVER TARGET  TO DB DB2C;
SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY SCHEMA ASN;

CREATE REPLQMAP "RQMA2C" USING
ADMINQ  "CAPA.ADMINQ.REMOTE"
RECVQ   "CAPA.TO.APPC.RECVQ"
SENDQ   "CAPA.TO.APPC.SENDQ.REMOTE";
```

From CLP-A, run the file as:

```
$ asnclp -f SYSA_crt_rqma2c.asnclp
```

Creating a Replication Queue Map for DB2B to DB2A

Follow the instructions in the *Bidirectional replication – The Q replication layer – Creating a Replication Queue Map for DB2B to DB2A* section.

Creating a Replication Queue Map for DB2B to DB2C

The Replication Queue Map for the queues going from `DB2B` to `DB2C` can be created using the following ASNCLP commands in the `SYSB_crt_rqmb2c.asnclp` file:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2B;
SET SERVER TARGET  TO DB DB2C;
SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY SCHEMA ASN;

CREATE REPLQMAP "RQMB2C" USING
ADMINQ  "CAPB.ADMINQ.REMOTE"
RECVQ   "CAPB.TO.APPC.RECVQ"
SENDQ   "CAPB.TO.APPC.SENDQ.REMOTE";
```

From CLP-B, run the file as:

```
$ asnclp -f SYSB_crt_rqmb2c.asnclp
```

Creating a Replication Queue Map for DB2C to DB2A

The Replication Queue Map for the queues going from `DB2C` to `DB2A` can be created using the following ASNCLP commands in the `SYSC_crt_rqmc2a.asnclp` file:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2C;
SET SERVER TARGET  TO DB DB2A;
SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY SCHEMA ASN;

CREATE REPLQMAP "RQMC2A" USING
ADMINQ  "CAPC.ADMINQ.REMOTE"
RECVQ   "CAPC.TO.APPA.RECVQ"
SENDQ   "CAPC.TO.APPA.SENDQ.REMOTE";
```

From CLP-C, run the file as:

```
$ asnclp -f SYSC_crt_rqmc2a.asnclp
```

Creating a Replication Queue Map for DB2C to DB2B

The Replication Queue Map for the queues going from `DB2C` to `DB2B` can be created using the following ASNCLP commands in the `SYSC_crt_rqmc2b.asnclp` file:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2C;
SET SERVER TARGET  TO DB DB2B;
SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY SCHEMA ASN;

CREATE REPLQMAP "RQMC2B" USING
ADMINQ  "CAPC.ADMINQ.REMOTE"
RECVQ   "CAPC.TO.APPB.RECVQ"
SENDQ   "CAPC.TO.APPB.SENDQ.REMOTE";
```

From CLP-C, run the file as:

```
$ asnclp -f SYSC_crt_rqmc2b.asnclp
```

Creating a Q subscription

The `SYSA_loadp2p3w.asnclp` load file runs the content `SYSA_contp2p3w.txt`, and contains:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

LOAD MULTIDIR REPL SCRIPT "SYSA_contp2p3w.txt";
The content file SYSA_contp2p3w.txt contains:
set subgroup "3Node0";

set output multidir;
set server multidir to db DB2A;
set multidir schema DB2A.ASN;
set server multidir to db DB2B;
set multidir schema DB2B.ASN;
set server multidir to db DB2C;
set multidir schema DB2C.ASN;
```

```
SET CONNECTION SOURCE DB2A.ASN TARGET DB2B.ASN REPLQMAP RQMA2B;
SET CONNECTION SOURCE DB2A.ASN TARGET DB2C.ASN REPLQMAP RQMA2C;
SET CONNECTION SOURCE DB2B.ASN TARGET DB2A.ASN REPLQMAP RQMB2A;
SET CONNECTION SOURCE DB2B.ASN TARGET DB2C.ASN REPLQMAP RQMB2C;
SET CONNECTION SOURCE DB2C.ASN TARGET DB2A.ASN REPLQMAP RQMC2A;
SET CONNECTION SOURCE DB2C.ASN TARGET DB2B.ASN REPLQMAP RQMC2B;

set tables(DB2A.ASN.ERIC.T1,DB2B.ASN.FRED.T1,DB2C.ASN.HEAT.T1);
CREATE QSUB subtype p;
```

From CLP-A, run the file as:

```
$ asnclp -f SYSA_loadp2p3w.asnclp
```

Starting Q Capture and Q Apply

Now we need to start Q Capture and Q Apply.

Starting Q Capture on DB2A, DB2B, and DB2C

To start Q Capture follow the instructions in the *Q Capture administration – Starting Q Capture* section of *Chapter 6*.

Wait for all Q Captures to be up and running before starting the Q Applys.

Starting Q Apply on DB2A, DB2B, and DB2C

To start Q Capture, follow the instructions in the *Q Apply administration – Starting Q Apply* section of *Chapter 6*.

Issuing a CAPSTART command

We now have to start the other Q subscriptions after we have started all the Q Captures and Q Applys.

We need to wait for the `T10001` and `T10004` Q subscriptions to be active before issuing the `CAPSTART` command. If we do not wait for this, then we will see the following message in the Q Capture log for `DB2A`:

```
<queueSub::findActiveP2PMember> ASN7063E  "Q
Capture" :  "ASN" : "WorkerThread" : Q subscription
"T10002" was not activated because another Q
subscription "T10001", which shares the same Q
subscription group, is in the process of being
activated.
```

If we get this message, then we need to check the `IBMQREP_SUBS` table to find out which Q subscriptions are in `I` state and reactivate them. We might first have to deactivate the Q subscription for `T10001` and then activate it again before continuing with the activation of the other Q subscriptions.

To activate the other Q subscriptions, we will use the ASNCLP commands in the `SYSA_qsub_start_db2ac.asnclp` file:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY SCHEMA ASN;

SET SERVER CAPTURE TO DB DB2A;
SET SERVER TARGET  TO DB DB2C;

START QSUB SUBNAME T10002;
```

From CLP-A, run the file as:

```
$ asnclp -f SYSA_qsub_start_db2ac.asnclp
```

Testing replication

We are now in a position to test our P2P three-way replication setup. We can insert a record into `ERIC.T1` on `DB2A` and check that it is replicated to `FRED.T1` on `DB2B` and `DB2C`.

From CLP-A, issue:

```
$ db2 "insert into eric.t1(c1,c2,c3) values (1,1,'J')"
```

From CLP-B, issue:

```
$ db2 "select * from fred.t1"
```

We should see one record in `FRED.T1` on `DB2B`.

From CLP-C, issue:

```
$ db2 "select * from heat.t1"
```

We should see one record in HEAT.T1 on DB2C.

And then if we insert a record into HEAT.T1 on DB2C and check that it is replicated to ERIC.T1 on DB2A and FRED.T1 on DB2B.

From CLP-C, issue:

```
$ db2 "insert into heat.t1(c1,c2,c3) values (2,2,'H')"
```

From CLP-A, issue:

```
$ db2 "select * from eric.t1"
```

We should see two records in ERIC.T1 on DB2A.

From CLP-B, issue:

```
$ db2 "select * from fred.t1"
```

We should see two records in FRED.T1 on DB2C.

And then finally if we insert a record into FRED.T1 on DB2B and check that it is replicated to ERIC.T1 on DB2A and HEAT.T1 on DB2C.

From CLP-B, issue:

```
$ db2 "insert into fred.t1(c1,c2,c3) values (3,3,'S')"
```

From CLP-A, issue:

```
$ db2 "select * from eric.t1"
```

We should see three records in ERIC.T1 on DB2A.

From CLP-C, issue:

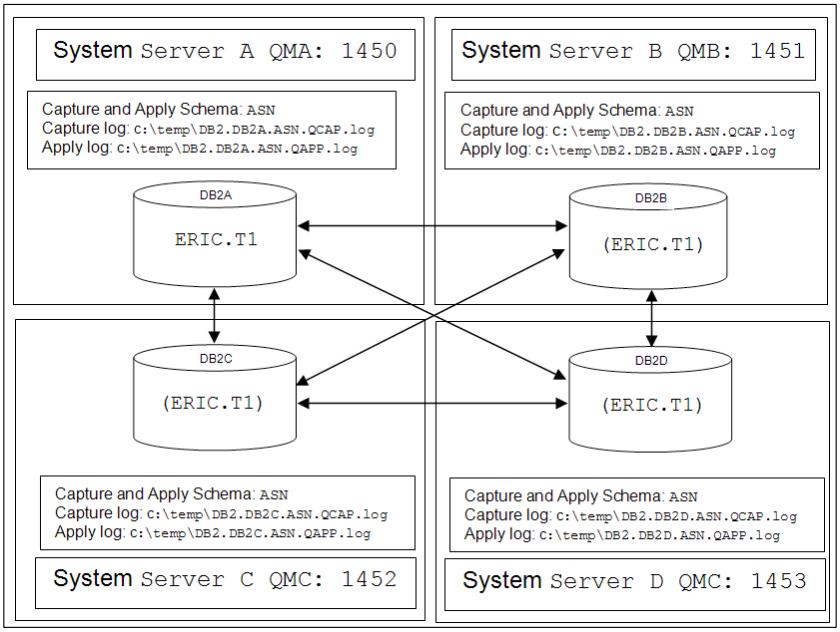
```
$ db2 "select * from heat.t1"
```

We should see three records in HEAT.T1 on DB2C.

We can see that the P2P three-way Q replication setup is working.

P2P four-way replication

The setup we will use is shown in the following diagram:



The test table `ERIC.T1` only exists on `DB2A`—Q Apply will create it on `DB2B`, `DB2C`, and `DB2D`.

The database layer

We need to create four source databases called `DB2A`, `DB2B`, `DB2C`, and `DB2D`—follow the instructions in the *First steps – Database creation* section.

We now have the database layer defined and can proceed to the WebSphere MQ layer.

The WebSphere MQ layer

This section deals with the WebSphere MQ layer, which covers creating the Queue Managers and the appropriate queues, and then starting the Listeners and Channels.

Creating the Queue Managers and Queues

We need to create and start four Queue Managers called QMA, QMB, QMC, and QMD—follow the instructions in the *First steps – Queue Manager processing* section.

The queues we need for P2P four-way replication are shown next.

The queues we need for QMA are in SYSA_QMA_MQDEFS_P2P4W_ABCD.TXT file:

```
DELETE QLOCAL(CAPA.ADMINQ) PURGE

DEFINE QLOCAL(CAPA.ADMINQ) +

REPLACE +

DESCR('LOCAL DEFN OF ADMINQ FOR
CAPA CAPTURE') +

PUT(ENABLED) +

GET(ENABLED) +

SHARE +

DEFSOPT(SHARED) +

DEFPSIST(YES)

*

DELETE QLOCAL(CAPA.RESTARTQ)
PURGE

DEFINE QLOCAL(CAPA.RESTARTQ) +

REPLACE +

DESCR('LOCAL DEFN OF RESTART FOR
CAPA CAPTURE') +

PUT(ENABLED) +

GET(ENABLED) +

SHARE +

DEFSOPT(SHARED) +

DEFPSIST(YES)

*

DEFINE QMODEL(IBMQREP.SPILL.
MODELQ) +

DEFINE QLOCAL(CAPC.TO.APPA.
RECVQ) +

REPLACE +

DESCR('LOCAL RECEIVE QUEUE -
APPA FROM CAPC') +

PUT(ENABLED) +

GET(ENABLED) +

DEFSOPT(SHARED) +

DEFPSIST(YES)

*

DEFINE QREMOTE(CAPC.ADMINQ.
REMOTE) +

REPLACE +

DESCR('REMOTE DEFN OF ADMINQ FOR
CAPC CAPTURE') +

PUT(ENABLED) +

XMITQ(QMC.XMITQ) +

RNAME(CAPC.ADMINQ) +

RQMNAME(QMC) +

DEFPSIST(YES)

*

DEFINE QLOCAL(QMC.XMITQ) +

REPLACE +

DESCR('TRANSMISSION QUEUE TO
QMC') +
```

```
REPLACE +
DEFSOPT(SHARED) +
MAXDEPTH(500000) +
MSGDLVSQ(FIFO) +
DEFTYPE(PERMDYN)
*
DEFINE QLOCAL(DEAD.LETTER.QUEUE.
QMA) +
REPLACE +
DESCR('LOCAL DEAD LETTER QUEUE
QMA') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QREMOTE(CAPA.TO.APPB.
SENDQ.REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPA TO APPB') +
PUT(ENABLED) +
XMITQ(QMB.XMITQ)  +
RNAME(CAPA.TO.APPB.RECVQ) +
RQMNAME(QMB) +
DEFPSIST(YES)
*
```

```
USAGE(XMITQ) +
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
TRIGTYPE(FIRST) +
TRIGDATA(QMA.TO.QMC) +
INITQ(SYSTEM.CHANNEL.INITQ)
*
DEFINE CHANNEL(QMA.TO.QMC) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMC') +
XMITQ(QMC.XMITQ) +
CONNAME('127.0.0.1(1452)')
*
DEFINE CHANNEL(QMC.TO.QMA) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM
QMC')
*
DEFINE QREMOTE(CAPA.TO.APPD.
SENDQ.REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPA TO APPD') +
```

```
DEFINE QLOCAL(CAPB.TO.APPA.RECVQ)
+
REPLACE +
DESCR('LOCAL RECEIVE QUEUE - APPA
FROM CAPB') +
PUT(ENABLED) +
GET(ENABLED) +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QREMOTE(CAPB.ADMINQ.
REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF ADMINQ FOR
CAPB CAPTURE') +
PUT(ENABLED) +
XMITQ(QMB.XMITQ) +
RNAME(CAPB.ADMINQ) +
RQMNAME(QMB) +
DEFPSIST(YES)
*
DEFINE QLOCAL(QMB.XMITQ) +
REPLACE +
DESCR('TRANSMISSION QUEUE TO
QMB') +
USAGE(XMITQ) +
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
TRIGTYPE(FIRST) +
PUT(ENABLED) +
XMITQ(QMD.XMITQ)  +
RNAME(CAPA.TO.APPD.RECVQ) +
RQMNAME(QMD) +
DEFPSIST(YES)
*
DEFINE QLOCAL(CAPD.TO.APPA.
RECVQ) +
REPLACE +
DESCR('LOCAL RECEIVE QUEUE -
APPA FROM CAPD') +
PUT(ENABLED) +
GET(ENABLED) +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QREMOTE(CAPD.ADMINQ.
REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF ADMINQ
FOR CAPD CAPTURE') +
PUT(ENABLED) +
XMITQ(QMD.XMITQ) +
RNAME(CAPD.ADMINQ) +
RQMNAME(QMD) +
DEFPSIST(YES)
*
DEFINE QLOCAL(QMD.XMITQ) +
REPLACE +
```

```
TRIGDATA(QMA.TO.QMB) +
INITQ(SYSTEM.CHANNEL.INITQ)
*
DEFINE CHANNEL(QMA.TO.QMB) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMB') +
XMITQ(QMB.XMITQ) +
CONNAME('127.0.0.1(1451)')
*
DEFINE CHANNEL(QMB.TO.QMA) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM
QMB')
*
DEFINE QREMOTE(CAPA.TO.APPC.
SENDQ.REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPA TO APPC') +
PUT(ENABLED) +
XMITQ(QMC.XMITQ)  +
RNAME(CAPA.TO.APPC.RECVQ) +
RQMNAME(QMC) +
DEFPSIST(YES)
DESCR('TRANSMISSION QUEUE TO
QMD') +
USAGE(XMITQ) +
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
TRIGTYPE(FIRST) +
TRIGDATA(QMA.TO.QMD) +
INITQ(SYSTEM.CHANNEL.INITQ)
*
DEFINE CHANNEL(QMA.TO.QMD) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMD')
+
XMITQ(QMD.XMITQ) +
CONNAME('127.0.0.1(1453)')
*
DEFINE CHANNEL(QMD.TO.QMA) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM
QMD')*
```

From CLP-A, run the file as:

```
$ runmqsc QMA < SYSA_QMA_MQDEFS_P2P4W_ABCD.TXT
```

The queues we need for `QMB` are in `SYSB_QMB_MQDEFS_P2P4W_ABCD.TXT` file:

```
DELETE QLOCAL(CAPB.ADMINQ) PURGE

DEFINE QLOCAL(CAPB.ADMINQ) +

REPLACE +

DESCR('LOCAL DEFN OF ADMINQ FOR
CAPB CAPTURE') +

PUT(ENABLED) +

GET(ENABLED) +

SHARE +

DEFSOPT(SHARED) +

DEFPSIST(YES)

*

DELETE QLOCAL(CAPB.RESTARTQ)
PURGE

DEFINE QLOCAL(CAPB.RESTARTQ) +

REPLACE +

DESCR('LOCAL DEFN OF RESTART FOR
CAPB CAPTURE') +

PUT(ENABLED) +

GET(ENABLED) +

SHARE +

DEFSOPT(SHARED) +

DEFPSIST(YES)

*

DEFINE QMODEL(IBMQREP.SPILL.
MODELQ) +

REPLACE +

DEFINE QLOCAL(CAPC.TO.APPB.
RECVQ) +

REPLACE +

DESCR('LOCAL RECEIVE QUEUE -
APPB FROM CAPC') +

PUT(ENABLED) +

GET(ENABLED) +

DEFSOPT(SHARED) +

DEFPSIST(YES)

*

DEFINE QREMOTE(CAPC.ADMINQ.
REMOTE) +

REPLACE +

DESCR('REMOTE DEFN OF ADMINQ FOR
CAPC CAPTURE') +

PUT(ENABLED) +

XMITQ(QMC.XMITQ) +

RNAME(CAPC.ADMINQ) +

RQMNAME(QMC) +

DEFPSIST(YES)

*

DEFINE QLOCAL(QMC.XMITQ) +

REPLACE +

DESCR('TRANSMISSION QUEUE TO
QMC') +

USAGE(XMITQ) +
```

```
DEFSOPT(SHARED) +
MAXDEPTH(500000) +
MSGDLVSQ(FIFO) +
DEFTYPE(PERMDYN)
*
DEFINE QLOCAL(DEAD.LETTER.QUEUE.
QMB) +
REPLACE +
DESCR('LOCAL DEAD LETTER QUEUE
QMB') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QREMOTE(CAPB.TO.APPA.
SENDQ.REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPB TO APPA') +
PUT(ENABLED) +
XMITQ(QMA.XMITQ)  +
RNAME(CAPB.TO.APPA.RECVQ) +
RQMNAME(QMA) +
DEFPSIST(YES)
*
DEFINE QLOCAL(CAPA.TO.APPB.RECVQ)
+
REPLACE +
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
TRIGTYPE(FIRST) +
TRIGDATA(QMB.TO.QMC) +
INITQ(SYSTEM.CHANNEL.INITQ)
*
DEFINE CHANNEL(QMB.TO.QMC) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMC') +
XMITQ(QMC.XMITQ) +
CONNAME('127.0.0.1(1452)')
*
DEFINE CHANNEL(QMC.TO.QMB) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM
QMC')
*
DEFINE QREMOTE(CAPB.TO.APPD.
SENDQ.REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPB TO APPD') +
PUT(ENABLED) +
```

```
DESCR('LOCAL RECEIVE QUEUE - APPB
FROM CAPA') +
PUT(ENABLED) +
GET(ENABLED) +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QREMOTE(CAPA.ADMINQ.
REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF ADMINQ FOR
CAPA CAPTURE') +
PUT(ENABLED) +
XMITQ(QMA.XMITQ) +
RNAME(CAPA.ADMINQ) +
RQMNAME(QMA) +
DEFPSIST(YES)
*
DEFINE QLOCAL(QMA.XMITQ) +
REPLACE +
DESCR('TRANSMISSION QUEUE TO
QMA') +
USAGE(XMITQ) +
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
TRIGTYPE(FIRST) +
TRIGDATA(QMB.TO.QMA) +
INITQ(SYSTEM.CHANNEL.INITQ)
*
XMITQ(QMD.XMITQ)  +
RNAME(CAPB.TO.APPD.RECVQ) +
RQMNAME(QMD) +
DEFPSIST(YES)
*
DEFINE QLOCAL(CAPD.TO.APPB.
RECVQ) +
REPLACE +
DESCR('LOCAL RECEIVE QUEUE -
APPB FROM CAPD') +
PUT(ENABLED) +
GET(ENABLED) +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QREMOTE(CAPD.ADMINQ.
REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF ADMINQ FOR
CAPD CAPTURE') +
PUT(ENABLED) +
XMITQ(QMD.XMITQ) +
RNAME(CAPD.ADMINQ) +
RQMNAME(QMD) +
DEFPSIST(YES)
*
DEFINE QLOCAL(QMD.XMITQ) +
REPLACE +
DESCR('TRANSMISSION QUEUE TO
QMD') +
```

```
DEFINE CHANNEL(QMB.TO.QMA) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMA') +
XMITQ(QMA.XMITQ) +
CONNAME('127.0.0.1(1450)')
*
DEFINE CHANNEL(QMA.TO.QMB) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM
QMA')
*
DEFINE QREMOTE(CAPB.TO.APPC.
SENDQ.REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPB TO APPC') +
PUT(ENABLED) +
XMITQ(QMC.XMITQ)  +
RNAME(CAPB.TO.APPC.RECVQ) +
RQMNAME(QMC) +
DEFPSIST(YES)  *
USAGE(XMITQ) +
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
TRIGTYPE(FIRST) +
TRIGDATA(QMB.TO.QMD) +
INITQ(SYSTEM.CHANNEL.INITQ)
*
DEFINE CHANNEL(QMB.TO.QMD) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMD') +
XMITQ(QMD.XMITQ) +
CONNAME('127.0.0.1(1453)')
*
DEFINE CHANNEL(QMD.TO.QMB) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM
QMD')
*
```

From CLP-B, run the file as:

```
$ runmqsc QMB < SYSB_QMB_MQDEFS_P2P4W_ABCD.TXT
```

The queues we need for `QMC` are in `SYSC_QMC_MQDEFS_P2P4W_ABCD.TXT` file:

```
DELETE QLOCAL(CAPC.ADMINQ) PURGE

DEFINE QLOCAL(CAPC.ADMINQ) +

REPLACE +

DESCR('LOCAL DEFN OF ADMINQ FOR
CAPC CAPTURE') +

PUT(ENABLED) +

GET(ENABLED) +

SHARE +

DEFSOPT(SHARED) +

DEFPSIST(YES)

*

DELETE QLOCAL(CAPC.RESTARTQ)
PURGE

DEFINE QLOCAL(CAPC.RESTARTQ) +

REPLACE +

DESCR('LOCAL DEFN OF RESTART FOR
CAPC CAPTURE') +

PUT(ENABLED) +

GET(ENABLED) +

SHARE +

DEFSOPT(SHARED) +

DEFPSIST(YES)

*

DEFINE QMODEL(IBMQREP.SPILL.
MODELQ) +

REPLACE +

DEFSOPT(SHARED) +

MAXDEPTH(500000) +

MSGDLVSQ(FIFO) +

DEFINE QREMOTE(CAPB.ADMINQ.
REMOTE) +

REPLACE +

DESCR('REMOTE DEFN OF ADMINQ FOR
CAPB CAPTURE') +

PUT(ENABLED) +

XMITQ(QMB.XMITQ) +

RNAME(CAPB.ADMINQ) +

RQMNAME(QMB) +

DEFPSIST(YES)

*

DEFINE QLOCAL(QMA.XMITQ) +

REPLACE +

DESCR('TRANSMISSION QUEUE TO
QMA') +

USAGE(XMITQ) +

PUT(ENABLED) +

GET(ENABLED) +

TRIGGER +

TRIGTYPE(FIRST) +

TRIGDATA(QMC.TO.QMA) +

INITQ(SYSTEM.CHANNEL.INITQ)

*

DEFINE QLOCAL(QMB.XMITQ) +

REPLACE +

DESCR('TRANSMISSION QUEUE TO
QMB') +

USAGE(XMITQ) +

PUT(ENABLED) +
```

```
DEFTYPE(PERMDYN)

*

DEFINE QLOCAL(DEAD.LETTER.QUEUE.
QMC) +

REPLACE +

DESCR('LOCAL DEAD LETTER QUEUE

QMC') +

PUT(ENABLED) +

GET(ENABLED) +

SHARE +

DEFSOPT(SHARED) +

DEFPSIST(YES)

*

DEFINE QREMOTE(CAPC.TO.APPA.
SENDQ.REMOTE) +

REPLACE +

DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPC TO APPA') +

PUT(ENABLED) +

XMITQ(QMA.XMITQ)  +

RNAME(CAPC.TO.APPA.RECVQ) +

RQMNAME(QMA) +

DEFPSIST(YES)

*

DEFINE QLOCAL(CAPA.TO.APPC.RECVQ)
+

REPLACE +

DESCR('LOCAL RECEIVE QUEUE - APPC
FROM CAPA') +

GET(ENABLED) +

TRIGGER +

TRIGTYPE(FIRST) +

TRIGDATA(QMC.TO.QMB) +

INITQ(SYSTEM.CHANNEL.INITQ)

*

DEFINE CHANNEL(QMC.TO.QMB) +

CHLTYPE(SDR) +

REPLACE +

TRPTYPE(TCP) +

DISCINT(0) +

DESCR('SENDER CHANNEL TO QMB') +

XMITQ(QMB.XMITQ) +

CONNAME('127.0.0.1(1451)')

*

DEFINE CHANNEL(QMB.TO.QMC) +

CHLTYPE(RCVR) +

REPLACE +

TRPTYPE(TCP) +

DESCR('RECEIVER CHANNEL FROM
QMB')

*

DEFINE QREMOTE(CAPC.TO.APPD.
SENDQ.REMOTE) +

REPLACE +

DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPC TO APPD') +

PUT(ENABLED) +

XMITQ(QMD.XMITQ)  +
```

```
PUT(ENABLED) +
GET(ENABLED) +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QREMOTE(CAPA.ADMINQ.
REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF ADMINQ FOR
CAPA CAPTURE') +
PUT(ENABLED) +
XMITQ(QMA.XMITQ) +
RNAME(CAPA.ADMINQ) +
RQMNAME(QMA) +
DEFPSIST(YES)
*
DEFINE CHANNEL(QMC.TO.QMA) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMA') +
XMITQ(QMA.XMITQ) +
CONNAME('127.0.0.1(1450)')
*
DEFINE CHANNEL(QMA.TO.QMC) +
CHLTYPE(RCVR) +
REPLACE +
RNAME(CAPC.TO.APPD.RECVQ) +
RQMNAME(QMD) +
DEFPSIST(YES)
*
DEFINE QLOCAL(CAPD.TO.APPC.
RECVQ) +
REPLACE +
DESCR('LOCAL RECEIVE QUEUE -
APPC FROM CAPD') +
PUT(ENABLED) +
GET(ENABLED) +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QREMOTE(CAPD.ADMINQ.
REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF ADMINQ FOR
CAPD CAPTURE') +
PUT(ENABLED) +
XMITQ(QMD.XMITQ) +
RNAME(CAPD.ADMINQ) +
RQMNAME(QMD) +
DEFPSIST(YES)
*
DEFINE QLOCAL(QMD.XMITQ) +
REPLACE +
DESCR('TRANSMISSION QUEUE TO
QMD') +
USAGE(XMITQ) +
```

```
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM
QMA')
*
DEFINE QREMOTE(CAPC.TO.APPB.
SENDQ.REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPC TO APPB') +
PUT(ENABLED) +
XMITQ(QMB.XMITQ)  +
RNAME(CAPC.TO.APPB.RECVQ) +
RQMNAME(QMB) +
DEFPSIST(YES)
*
DEFINE QLOCAL(CAPB.TO.APPC.
RECVQ) +
REPLACE +
DESCR('LOCAL RECEIVE QUEUE -
APPC FROM CAPB') +
PUT(ENABLED) +
GET(ENABLED) +
DEFSOPT(SHARED) +
DEFPSIST(YES)*
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
TRIGTYPE(FIRST) +
TRIGDATA(QMC.TO.QMD) +
INITQ(SYSTEM.CHANNEL.INITQ)
*
DEFINE CHANNEL(QMC.TO.QMD) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMD') +
XMITQ(QMD.XMITQ) +
CONNAME('127.0.0.1(1453)')
*
DEFINE CHANNEL(QMD.TO.QMC) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM
QMD')*
```

From CLP-C, run the file as:

```
$ runmqsc QMC < SYSC_QMC_MQDEFS_P2P4W_ABCD.TXT
```

The queues we need for `QMD` are in `SYSD_QMD_MQDEFS_P2P4W_ABCD.TXT` file:

```
DELETE QLOCAL(CAPD.ADMINQ) PURGE

DEFINE QLOCAL(CAPD.ADMINQ) +

REPLACE +

DESCR('LOCAL DEFN OF ADMINQ FOR
CAPD CAPTURE') +

PUT(ENABLED) +

GET(ENABLED) +

SHARE +

DEFSOPT(SHARED) +

DEFPSIST(YES)

*

DELETE QLOCAL(CAPD.RESTARTQ)
PURGE

DEFINE QLOCAL(CAPD.RESTARTQ) +

REPLACE +

DESCR('LOCAL DEFN OF RESTART FOR
CAPD CAPTURE') +

PUT(ENABLED) +

GET(ENABLED) +

SHARE +

DEFSOPT(SHARED) +

DEFPSIST(YES)

*

DEFINE QMODEL(IBMQREP.SPILL.
MODELQ) +

REPLACE +

DEFSOPT(SHARED) +

MAXDEPTH(500000) +

MSGDLVSQ(FIFO) +

DEFTYPE(PERMDYN)

*

DEFINE QLOCAL(CAPB.TO.APPD.
RECVQ) +

REPLACE +

DESCR('LOCAL RECEIVE QUEUE -
APPD FROM CAPB') +

PUT(ENABLED) +

GET(ENABLED) +

DEFSOPT(SHARED) +

DEFPSIST(YES)

*

DEFINE QREMOTE(CAPB.ADMINQ.
REMOTE) +

REPLACE +

DESCR('REMOTE DEFN OF ADMINQ FOR
CAPB CAPTURE') +

PUT(ENABLED) +

XMITQ(QMB.XMITQ) +

RNAME(CAPB.ADMINQ) +

RQMNAME(QMB) +

DEFPSIST(YES)

*

DEFINE QLOCAL(QMB.XMITQ) +

REPLACE +

DESCR('TRANSMISSION QUEUE TO
QMB') +

USAGE(XMITQ) +

PUT(ENABLED) +

GET(ENABLED) +

TRIGGER +

TRIGTYPE(FIRST) +

TRIGDATA(QMD.TO.QMB) +
```

```
DEFINE QLOCAL(DEAD.LETTER.QUEUE.
QMD) +
REPLACE +
DESCR('LOCAL DEAD LETTER QUEUE
QMD') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QREMOTE(CAPD.TO.APPA.
SENDQ.REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPD TO APPA') +
PUT(ENABLED) +
XMITQ(QMA.XMITQ)  +
RNAME(CAPD.TO.APPA.RECVQ) +
RQMNAME(QMA) +
DEFPSIST(YES)
*
DEFINE QLOCAL(CAPA.TO.APPD.RECVQ)
+
REPLACE +
DESCR('LOCAL RECEIVE QUEUE - APPD
FROM CAPA') +
PUT(ENABLED) +
GET(ENABLED) +
DEFSOPT(SHARED) +
DEFPSIST(YES)
INITQ(SYSTEM.CHANNEL.INITQ)
*
DEFINE CHANNEL(QMD.TO.QMB) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMB') +
XMITQ(QMB.XMITQ) +
CONNAME('127.0.0.1(1451)')
*
DEFINE CHANNEL(QMB.TO.QMD) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM
QMB')
*
DEFINE QREMOTE(CAPD.TO.APPC.
SENDQ.REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPD TO APPC') +
PUT(ENABLED) +
XMITQ(QMC.XMITQ)  +
RNAME(CAPD.TO.APPC.RECVQ) +
RQMNAME(QMC) +
DEFPSIST(YES)
*
DEFINE QLOCAL(CAPC.TO.APPD.
RECVQ) +
```

```
*
DEFINE QREMOTE(CAPA.ADMINQ.
REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF ADMINQ FOR
CAPA CAPTURE') +
PUT(ENABLED) +
XMITQ(QMA.XMITQ) +
RNAME(CAPA.ADMINQ) +
RQMNAME(QMA) +
DEFPSIST(YES)
*
DEFINE QLOCAL(QMA.XMITQ) +
REPLACE +
DESCR('TRANSMISSION QUEUE TO
QMA') +
USAGE(XMITQ) +
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
TRIGTYPE(FIRST) +
TRIGDATA(QMD.TO.QMA) +
INITQ(SYSTEM.CHANNEL.INITQ)
*
DEFINE CHANNEL(QMD.TO.QMA) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
DISCINT(0) +
```

```
+
REPLACE +
DESCR('LOCAL RECEIVE QUEUE -
APPD FROM CAPC') +
PUT(ENABLED) +
GET(ENABLED) +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QREMOTE(CAPC.ADMINQ.
REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF ADMINQ FOR
CAPC CAPTURE') +
PUT(ENABLED) +
XMITQ(QMC.XMITQ) +
RNAME(CAPC.ADMINQ) +
RQMNAME(QMC) +
DEFPSIST(YES)
*
DEFINE QLOCAL(QMC.XMITQ) +
REPLACE +
DESCR('TRANSMISSION QUEUE TO
QMC') +
USAGE(XMITQ) +
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
TRIGTYPE(FIRST) +
TRIGDATA(QMD.TO.QMC) +
```

```
DESCR('SENDER CHANNEL TO QMA') +
XMITQ(QMA.XMITQ) +
CONNAME('127.0.0.1(1450)')
*
DEFINE CHANNEL(QMA.TO.QMD) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM
QMA')
*
DEFINE QREMOTE(CAPD.TO.APPB.
SENDQ.REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPD TO APPB') +
PUT(ENABLED) +
XMITQ(QMB.XMITQ)  +
RNAME(CAPD.TO.APPB.RECVQ) +
RQMNAME(QMB) +
DEFPSIST(YES)  *
INITQ(SYSTEM.CHANNEL.INITQ)
*
DEFINE CHANNEL(QMD.TO.QMC) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMC')
+
XMITQ(QMC.XMITQ) +
CONNAME('127.0.0.1(1452)')
*
DEFINE CHANNEL(QMC.TO.QMD) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM
QMC')
```

From CLP-D, run the file as:

```
$ runmqsc QMD < SYSB_QMD_MQDEFS_P2P4W_ABCD.TXT
```

Now that we have defined all the queues, we can start the Listeners and Channels.

Starting the Listeners

Start the Listeners for `QMA`, `QMB`, and `QMC` as described in the *Bidirectional replication – The WebSphere MQ layer – Starting the Listeners* section.

The Listener for QMD can be started using the SYSC_QMD_START_RUNMQLSR.BAT batch file as shown next:

```
start runmqlsr -t tcp -m  QMD   -p  1453
```

From CLP-D, run the file as:

```
$ SYSC_QMD_START_RUNMQLSR.BAT
```

Starting the Channels

Start the Channels between QMA, QMB, and QMC as described in the *P2P three-way replication – The WebSphere MQ layer – Starting the Channels* section.

The Channel from QMA to QMD can be started by executing the contents of the SYSA_QMA_START_RUNMQCHL_AD.BAT file:

```
start runmqchl -m  QMA -c QMA.TO.QMD
```

From CLP-A, run the file as:

```
$ SYSA_QMA_START_RUNMQCHL_AD.BAT
```

The Channel from QMB to QMD can be started by executing the contents of the SYSB_QMB_START_RUNMQCHL_BD.BAT file:

```
start runmqchl -m  QMB -c QMB.TO.QMD
```

From CLP-B, run the file as:

```
$ SYSB_QMB_START_RUNMQCHL_BD.BAT
```

The Channel from QMC to QMD can be started by executing the contents of the SYSC_QMC_START_RUNMQCHL_CD.BAT file:

```
start runmqchl -m  QMC -c QMC.TO.QMD
```

From CLP-C, run the file as:

```
$ SYSC_QMC_START_RUNMQCHL_CD.BAT
```

The Channel from QMD to QMA can be started by executing the contents of the SYSC_QMC_START_RUNMQCHL_DA.BAT file:

```
start runmqchl -m  QMD -c QMD.TO.QMA
```

The Channel from QMD to QMB can be started by executing the contents of the SYSD_QMD_START_RUNMQCHL_DB.BAT file:

```
start runmqchl -m  QMD -c QMD.TO.QMB
```

The Channel from QMD to QMC can be started by executing the contents of the SYSD_QMD_START_RUNMQCHL_DC.BAT file:

```
start runmqchl -m  QMD -c QMD.TO.QMC
```

From CLP-D, run the files as:

```
$ SYSD_QMD_START_RUNMQCHL_DB.BAT
$ SYSD_QMD_START_RUNMQCHL_DA.BAT
$ SYSD_QMD_START_RUNMQCHL_DC.BAT
```

Testing the WebSphere MQ layer

Now that everything is started, we need to test the MQ layer.

We will start by putting test messages onto each system using the amqsput command and then retrieving them using the amqsget command.

The put message batch file for QMA is called SYSA_QMA_TESTP_P2P4W.BAT:

```
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPA.TO.APPB.
SENDQ.REMOTE     QMA < SYSA_QMA_TEST1.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPB.ADMINQ.
REMOTE     QMA < SYSA_QMA_TEST2.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPA.TO.APPC.
SENDQ.REMOTE     QMA < SYSA_QMA_TEST3.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPC.ADMINQ.
REMOTE     QMA < SYSA_QMA_TEST4.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPA.TO.APPD.
SENDQ.REMOTE     QMA < SYSA_QMA_TEST5.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPD.ADMINQ.
REMOTE     QMA < SYSA_QMA_TEST6.TXT
```

From CLP-A, run the file as:

```
$ SYSA_QMA_TESTP_P2P4W.BAT
```

The put message batch file for QMB is called SYSB_QMB_TESTP_P2P4W.BAT:

```
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPB.TO.APPA.
SENDQ.REMOTE     QMB < SYSB_QMB_TEST7.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPA.ADMINQ.
REMOTE     QMB < SYSB_QMB_TEST8.TXT
```

```
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPB.TO.APPC.
SENDQ.REMOTE     QMB < SYSB_QMB_TEST9.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPC.ADMINQ.
REMOTE     QMB < SYSB_QMB_TEST10.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPB.TO.APPD.
SENDQ.REMOTE     QMB < SYSB_QMB_TEST11.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPD.ADMINQ.
REMOTE     QMB < SYSB_QMB_TEST12.TXT
```

From CLP-B, run the file as:

```
$ SYSB_QMB_TESTP_P2P4W.BAT
```

The put message batch file for `QMC` is called `SYSC_QMC_TESTP_P2P4W.BAT`:

```
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPC.TO.APPA.
SENDQ.REMOTE     QMC < SYSC_QMC_TEST13.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPA.ADMINQ.
REMOTE     QMC < SYSC_QMC_TEST14.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPC.TO.APPB.
SENDQ.REMOTE     QMC < SYSC_QMC_TEST15.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPB.ADMINQ.
REMOTE     QMC < SYSC_QMC_TEST16.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPC.TO.APPD.
SENDQ.REMOTE     QMC < SYSC_QMC_TEST17.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPD.ADMINQ.
REMOTE     QMC < SYSC_QMC_TEST18.TXT
```

From CLP-C, run the file as:

```
$ SYSC_QMC_TESTP_P2P4W.BAT
```

The put message batch file for `QMD` is called `SYSD_QMD_TESTP_P2P4W.BAT`:

```
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPD.TO.APPA.
SENDQ.REMOTE     QMD < SYSD_QMD_TEST19.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPA.ADMINQ.
REMOTE     QMD < SYSD_QMD_TEST20.TXT
```

```
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPD.TO.APPB.
SENDQ.REMOTE     QMD < SYSD_QMD_TEST21.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPB.ADMINQ.
REMOTE     QMD < SYSD_QMD_TEST22.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPD.TO.APPC.
SENDQ.REMOTE     QMD < SYSD_QMD_TEST23.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPC.ADMINQ.
REMOTE     QMD < SYSD_QMD_TEST24.TXT
```

From CLP-D, run the file as:

```
$ SYSD_QMD_TESTP_P2P4W.BAT
```

Once we have put the test messages onto each system, we can retrieve them.

The get message batch file for `QMA` is called `SYSA_QMA_TESTG_P2P4W.BAT`:

```
@echo The amqsget program take 15 seconds to run

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPA.ADMINQ
QMA

@ECHO You should see above:  test8 test14 test20

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPB.TO.APPA.
RECVQ QMA

@ECHO You should see above:  test7

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPC.TO.APPA.
RECVQ QMA

@ECHO You should see above:  test13

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPD.TO.APPA.
RECVQ QMA

@ECHO You should see above:  test19
```

From CLP-A, run the file as:

```
$ SYSA_QMA_TESTG_P2P4W.BAT
```

The get message batch file for `QMB` is called `SYSB_QMB_TESTG_P2P4W.BAT`:

```
@echo The amqsget program take 15 seconds to run

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPA.TO.APPB.
RECVQ QMB

@ECHO You should see above:  test1
```

```
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPB.ADMINQ
QMB

@ECHO You should see above:  test2 test16 test22

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPC.TO.APPB.
RECVQ QMB

@ECHO You should see above:  test15

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPD.TO.APPB.
RECVQ QMB

@ECHO You should see above:  test21
```

From CLP-B, run the file as:

```
$ SYSB_QMB_TESTG_P2P4W.BAT
```

The get message batch file for `QMC` is called `SYSC_QMC_TESTG_P2P4W.BAT`:

```
echo The amqsget program take 15 seconds to run

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPA.TO.APPC.
RECVQ QMC

@ECHO You should see above:  test3

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPB.TO.APPC.
RECVQ QMC

@ECHO You should see above:  test9

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPC.ADMINQ
QMC

@ECHO You should see above:  test4 test10 test24

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPD.TO.APPC.
RECVQ QMC

@ECHO You should see above:  test23
```

From CLP-C, run the file as:

```
$ SYSC_QMC_TESTG_P2P4W.BAT
```

The get message batch file for `QMD` is called `SYSD_QMD_TESTG_P2P4W.BAT`:

```
echo The amqsget program take 15 seconds to run

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPA.TO.APPD.
RECVQ QMD

@ECHO You should see above:  test5
```

```
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPB.TO.APPD.
RECVQ QMD

@ECHO You should see above:  test11

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPC.TO.APPD.
RECVQ QMD

@ECHO You should see above:  test17

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPD.ADMINQ
QMD

@ECHO You should see above:  test6 test12 test18
```

From CLP-D, run the file as:

```
$ SYSD_QMD_TESTG_P2P4W.BAT
```

Provided we see the messages that we are told we should see, then we have successfully tested the WebSphere MQ layer.

We have now defined the database and WebSphere MQ layers, and can proceed to the Q replication layer.

The Q replication layer

The following sections give the `ASNCLP` commands to create the control tables, the Replication Queue Maps and the Q subscription. The tasks are:

- Creating the Q Capture and Q Apply control tables on `DB2A`
- Creating the Q Capture and Q Apply control tables on `DB2B`
- Creating the Q Capture and Q Apply control tables on `DB2C`
- Creating the Q Capture and Q Apply control tables on `DB2D`
- Creating a Replication Queue Map for `DB2A` to `DB2B`
- Creating a Replication Queue Map for `DB2A` to `DB2C`
- Creating a Replication Queue Map for `DB2A` to `DB2D`
- Creating a Replication Queue Map for `DB2B` to `DB2A`
- Creating a Replication Queue Map for `DB2B` to `DB2C`
- Creating a Replication Queue Map for `DB2B` to `DB2D`
- Creating a Replication Queue Map for `DB2C` to `DB2A`
- Creating a Replication Queue Map for `DB2C` to `DB2B`
- Creating a Replication Queue Map for `DB2C` to `DB2D`
- Creating a Replication Queue Map for `DB2D` to `DB2A`

- Creating a Replication Queue Map for `DB2D` to `DB2B`
- Creating a Replication Queue Map for `DB2D` to `DB2C`
- Creating a Q subscription

Creating Q Capture/Q Apply control tables on DB2A

Follow the instructions in the *Bidirectional replication – The Q replication layer – Creating Q Capture/Q Apply control tables on DB2A* section.

Creating Q Capture/Q Apply control tables on DB2B

Follow the instructions in the *Bidirectional replication – The Q replication layer – Creating Q Capture/Q Apply control tables on DB2B* section.

Creating Q Capture/Q Apply control tables on DB2C

Follow the instructions in the *P2P three-way replication – The Q replication layer – Creating Q Capture/Q Apply control tables on DB2C* section.

Creating Q Capture/Q Apply control tables on DB2D

The Q Capture and Q Apply control tables for `DB2D` can be created using the following ASNCLP commands in the `SYSD_db2d_crt_capture_apply.asnclp` file:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2D ;
SET SERVER TARGET  TO DB DB2D ;
SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY   SCHEMA        ASN;
SET QMANAGER QMD FOR CAPTURE SCHEMA;
SET QMANAGER QMD FOR APPLY   SCHEMA;

CREATE CONTROL TABLES FOR APPLY SERVER USING
MONITOR LIMIT 3
TRACE LIMIT 9;

CREATE CONTROL TABLES FOR CAPTURE SERVER USING
RESTARTQ "CAPD.RESTARTQ"
ADMINQ "CAPD.ADMINQ"
STARTMODE WARMSI
MEMORY LIMIT 4
MONITOR INTERVAL 10;
```

From CLP-D, run the file as:

```
$ asnclp -f SYSD_db2d_crt_capture_apply.asnclp
```

Creating a Replication Queue Map for DB2A to DB2B

Follow the instructions in the *Bidirectional replication – The Q replication layer – Creating a Replication Queue Map for DB2A to DB2B* section.

Creating a Replication Queue Map for DB2A to DB2C

Follow the instructions in the *P2P three-way replication – The Q replication layer – Creating a Replication Queue Map for DB2A to DB2C* section.

Creating a Replication Queue Map for DB2A to DB2D

The Replication Queue Map for the queues going from `DB2A` to `DB2D` can be created using the following ASNCLP commands in the `SYSA_crt_rqma2d.asnclp` file:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2A;
SET SERVER TARGET  TO DB DB2D;
SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY SCHEMA ASN;

CREATE REPLQMAP "RQMA2D" USING
ADMINQ  "CAPA.ADMINQ.REMOTE"
RECVQ   "CAPA.TO.APPD.RECVQ"
SENDQ   "CAPA.TO.APPD.SENDQ.REMOTE";
```

From CLP-A, run the file as:

```
$ asnclp -f SYSA_crt_rqma2d.asnclp
```

Creating a Replication Queue Map for DB2B to DB2A

Follow the instructions in the *P2P three-way replication – The Q replication layer – Creating a Replication Queue Map for DB2B to DB2A* section.

Creating a Replication Queue Map for DB2B to DB2C

Follow the instructions in the *P2P three-way replication – The Q replication layer – Creating a Replication Queue Map for DB2B to DB2C* section.

Creating a Replication Queue Map for DB2B to DB2D

The Replication Queue Map for the queues going from `DB2B` to `DB2C` can be created using the following ASNCLP commands in the `SYSB_crt_rqmb2d.asnclp` file:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2B;
SET SERVER TARGET  TO DB DB2D;
SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY SCHEMA ASN;

CREATE REPLQMAP "RQMB2D" USING
ADMINQ  "CAPB.ADMINQ.REMOTE"
RECVQ   "CAPB.TO.APPD.RECVQ"
SENDQ   "CAPB.TO.APPD.SENDQ.REMOTE";
```

From CLP-B, run the file as:

```
$ asnclp -f SYSB_crt_rqmb2d.asnclp
```

Creating a Replication Queue Map for DB2C to DB2A

Follow the instructions in the *P2P three-way replication – The Q replication layer – Creating a Replication Queue Map for DB2C to DB2A* section.

Creating a Replication Queue Map for DB2C to DB2B

Follow the instructions in the *P2P three-way replication – The Q replication layer – Creating a Replication Queue Map for DB2C to DB2B* section.

Creating a Replication Queue Map for DB2C to DB2D

The Replication Queue Map for the queues going from DB2C to DB2A can be created using the following ASNCLP commands in the SYSC_crt_rqmc2d.asnclp file:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2C;
SET SERVER TARGET  TO DB DB2D;
SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY SCHEMA ASN;

CREATE REPLQMAP "RQMC2D" USING
ADMINQ  "CAPC.ADMINQ.REMOTE"
RECVQ   "CAPC.TO.APPD.RECVQ"
SENDQ   "CAPC.TO.APPD.SENDQ.REMOTE";
```

From CLP-C, run the file as:

```
$ asnclp -f SYSC_crt_rqmc2d.asnclp
```

Creating a Replication Queue Map for DB2D to DB2A

The Replication Queue Map for the queues going from DB2D to DB2A can be created using the following ASNCLP commands in the SYSD_crt_rqmd2a.asnclp file:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2D;
SET SERVER TARGET  TO DB DB2A;
SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY SCHEMA ASN;

CREATE REPLQMAP "RQMD2A" USING
ADMINQ  "CAPD.ADMINQ.REMOTE"
RECVQ   "CAPD.TO.APPA.RECVQ"
SENDQ   "CAPD.TO.APPA.SENDQ.REMOTE";
```

From CLP-D, run the file as:

```
$ asnclp -f SYSD_crt_rqmd2a.asnclp
```

Creating a Replication Queue Map for DB2D to DB2B

The Replication Queue Map for the queues going from `DB2D` to `DB2B` can be created using the following ASNCLP commands in the `SYSD_crt_rqmd2b.asnclp` file:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2D;
SET SERVER TARGET  TO DB DB2B;
SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY SCHEMA ASN;

CREATE REPLQMAP "RQMD2B" USING
ADMINQ  "CAPD.ADMINQ.REMOTE"
RECVQ   "CAPD.TO.APPB.RECVQ"
SENDQ   "CAPD.TO.APPB.SENDQ.REMOTE";
```

From CLP-D, run the file as:

```
$ asnclp -f SYSD_crt_rqmd2b.asnclp
```

Creating a Replication Queue Map for DB2D to DB2C

The Replication Queue Map for the queues going from `DB2D` to `DB2C` can be created using the following ASNCLP commands in the `SYSD_crt_rqmd2c.asnclp` file:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2D;
SET SERVER TARGET  TO DB DB2C;
SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY SCHEMA ASN;

CREATE REPLQMAP "RQMD2C" USING
ADMINQ  "CAPD.ADMINQ.REMOTE"
RECVQ   "CAPD.TO.APPC.RECVQ"
SENDQ   "CAPD.TO.APPC.SENDQ.REMOTE";
```

From CLP-D, run the file as:

```
$ asnclp -f SYSD_crt_rqmd2c.asnclp
```

Creating a Q subscription

Here is the P2P four-way `SYSA_loadp2p4w.asnclp` load file:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

LOAD MULTIDIR REPL SCRIPT "SYSA_contp2p4w.txt";
```

This is the P2P four-way `SYSA_contp2p4w.txt` content file:

```
set subgroup "TABT1";

set server multidir to db "DB2A";
set server multidir to db "DB2B";
set server multidir to db "DB2C";
set server multidir to db "DB2D";

set multidir schema "DB2A".ASN;
set multidir schema "DB2B".ASN;
set multidir schema "DB2C".ASN;
set multidir schema "DB2D".ASN;

SET CONNECTION SOURCE DB2A.ASN TARGET DB2B.ASN REPLQMAP RQMA2B;
SET CONNECTION SOURCE DB2A.ASN TARGET DB2C.ASN REPLQMAP RQMA2C;
SET CONNECTION SOURCE DB2A.ASN TARGET DB2D.ASN REPLQMAP RQMA2D;
SET CONNECTION SOURCE DB2B.ASN TARGET DB2A.ASN REPLQMAP RQMB2A;
SET CONNECTION SOURCE DB2B.ASN TARGET DB2C.ASN REPLQMAP RQMB2C;
SET CONNECTION SOURCE DB2B.ASN TARGET DB2D.ASN REPLQMAP RQMB2D;
SET CONNECTION SOURCE DB2C.ASN TARGET DB2A.ASN REPLQMAP RQMC2A;
SET CONNECTION SOURCE DB2C.ASN TARGET DB2B.ASN REPLQMAP RQMC2B;
SET CONNECTION SOURCE DB2C.ASN TARGET DB2D.ASN REPLQMAP RQMC2D;
SET CONNECTION SOURCE DB2D.ASN TARGET DB2A.ASN REPLQMAP RQMD2A;
SET CONNECTION SOURCE DB2D.ASN TARGET DB2B.ASN REPLQMAP RQMD2B;
SET CONNECTION SOURCE DB2D.ASN TARGET DB2C.ASN REPLQMAP RQMD2C;

set tables("DB2A".ASN.ERIC.T1, "DB2B".ASN.ERIC.T1,
"DB2C".ASN.ERIC.T1, "DB2D".ASN.ERIC.T1);

CREATE QSUB subtype p;
```

If we wanted to change the name of the table on the other DB2 systems, then this is the place to do it.

From CLP-A, run the file as:

```
$ asnclp -f SYSA_loadp2p4w.asnclp
```

The following Q subscriptions are created

A	B	C	D
T10001	T10004	T10007	T10010
T10002	T10005	T10008	T10011
T10003	T10006	T10009	T10012

Starting Q Capture and Q Apply

Now we need to start Q Capture and Q Apply.

Starting Q Capture on DB2A, DB2B, DB2C, and DB2D

To start Q Capture, follow the instructions in the *Q Capture administration – Starting Q Capture* section of *Chapter 6.*

Wait for all Q Captures to be up and running before starting the Q Applys.

Starting Q Apply on DB2A, DB2B, DB2C, and DB2D

To start Q Capture, follow the instructions in the *Q Apply administration – Starting Q Apply* section of *Chapter 6*.

Let's check the status of each of the Q subscriptions in the Q subscription group using:

```
$ db2 "select substr(subname,1,10) as subname, state as S, state_time
from asn.ibmqrep_subs"
```

From CLP-A:

```
SUBNAME    S STATE_TIME
---------- - --------------------------
T10001     A 2006-02-22-18.32.22.112000
T10002     I 2006-02-22-18.22.14.468000
T10003     I 2006-02-22-18.22.14.478002
```

We can see that the status of `T10001` is `A`, which means active. This is set automatically when Q Capture and Q Apply start. This is the only one that is started automatically (along with its pair on `DB2B`)—we need to start the other Q subscriptions manually.

From CLP-B:

```
SUBNAME    S STATE_TIME
---------- - --------------------------
T10004     A 2006-02-22-18.31.45.399001
T10005     I 2006-02-22-18.22.15.960000
T10006     I 2006-02-22-18.22.15.970004
```

We can see that the status of `T10004` is `A`, which means active.

From CLP-C:

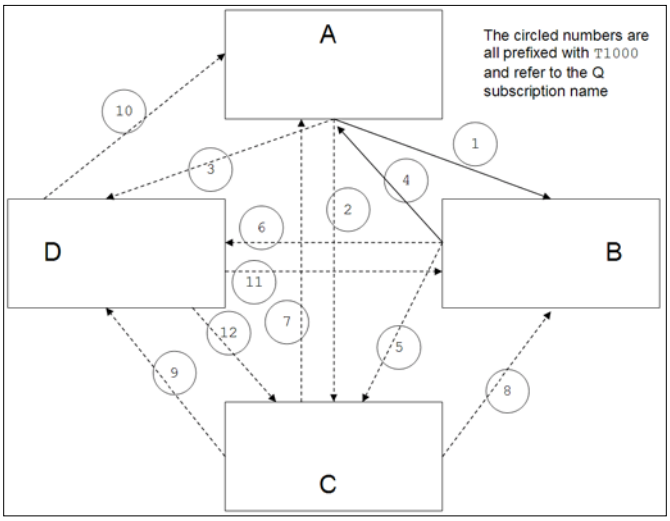
```
SUBNAME    S STATE_TIME
---------- - --------------------------
T10007     I 2006-02-22-18.22.17.663001
T10008     I 2006-02-22-18.22.17.683001
T10009     I 2006-02-22-18.22.17.693003
```

The status of all the Q subscriptions is `I` which means inactive—we will need to start these manually.

From CLP-D:

```
SUBNAME    S STATE_TIME
---------- - --------------------------
T10010     I 2006-02-22-18.22.19.185003
T10011     I 2006-02-22-18.22.19.205000
T10012     I 2006-02-22-18.22.19.235000
```

The status of all the Q subscriptions is `I` which means inactive—we will need to start these manually. This situation is shown in the following diagram, where the complete lines show active Q subscriptions, and the dotted lines show inactive Q subscriptions in the Q subscription group.



We see that the Q subscriptions `T10001` and `T10004` are the only ones that are activated. We now need to start the others manually.

Issuing CAPSTART command(s)

We need to issue a `CAPSTART` command.

We need to wait for the `T10001` and `T10004` Q subscriptions to be active before issuing the `CAPSTART` command. If we do not wait for this, then we will see the following in the Q Capture log for `DB2A`:

```
<queueSub::findActiveP2PMember> ASN7063E  "Q Capture"
:  "ASN" : "WorkerThread" : Q subscription "T10002" was
not activated because another Q subscription "T10001",
which shares the same Q subscription group, is in the
process of being activated.
```

If we get this message, then we need to check the `IBMQREP_SUBS` table to find out which Q subscriptions are in `I` state and reactivate them. We might first have to deactivate the Q subscription for `T10001` and then activate it again before continuing with the activation of the other Q subscriptions.

To issue the `CAPSTART` command, we will use the ASNCLP commands in the `SYSA_qsub_start_db2ac.asnclp` file:

From CLP-A:

```
$ asnclp -f SYSA_qsub_start_db2ac.asnclp
```

We can check the status of the Q subscription using the previous query.

From CLP-A:

```
SUBNAME    S STATE_TIME
---------- - --------------------------
T10001     A 2006-02-22-18.32.22.112000
T10002     A 2006-02-22-18.37.33.470001
T10003     I 2006-02-22-18.22.14.478002
```

From CLP-B:

```
SUBNAME    S STATE_TIME
---------- - --------------------------
T10004     A 2006-02-22-18.37.26.360002
T10005     A 2006-02-22-18.37.36.394000
T10006     I 2006-02-22-18.22.15.970004
```

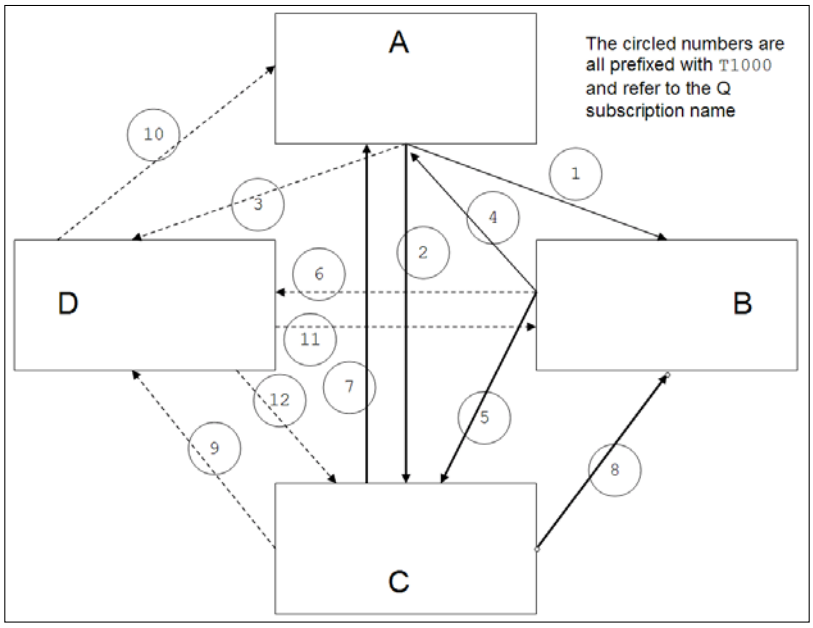
From CLP-C:

```
SUBNAME    S STATE_TIME
---------- - --------------------------
T10007     A 2006-02-22-18.37.23.546001
T10008     A 2006-02-22-18.37.33.580000
T10009     I 2006-02-22-18.22.17.693003
```

From CLP-D:

```
SUBNAME    S STATE_TIME
---------- - --------------------------
T10010     I 2006-02-22-18.22.19.185003
T10011     I 2006-02-22-18.22.19.205000
T10012     I 2006-02-22-18.22.19.235000
```

The following diagram shows the state of the Q subscriptions after issuing a `CAPSTART` command between `A` and `C` (the bold letters show the activated Q subscriptions).



We need to issue another `CAPSTART` command from `A` to `D`.

We need to wait for the `T10001`, `T10004`, `T10002`, `T10005`, `T10007`, and `T10008` Q subscriptions to be active before issuing the `CAPSTART`. If we do not wait for this, then we will see the `ASN7063E` message, discussed earlier, in the Q Capture log for `DB2A`.

To issue the second `CAPSTART` command, we will use the ASNCLP commands in the `SYSA_qsub_start_db2ad.asnclp` file:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2A;
SET SERVER TARGET  TO DB DB2D;

SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY SCHEMA ASN;

START QSUB SUBNAME T10003;
```

From CLP-A, issue:

```
$ asnclp -f SYSA_qsub_start_db2ad.asnclp
```

Let's check the status of the Q subscription again (using the previous query):

From CLP-A:

```
SUBNAME    S STATE_TIME
---------- - --------------------------
T10001     A 2006-02-22-18.32.22.112000
T10002     A 2006-02-22-18.37.33.470001
T10003     T 2006-02-22-18.43.24.374000
```

If we issue the command immediately after issuing the `CAPSTART` command, we will see a state of `T`.

Issue the command again after a few seconds and the `STATE(S)` should change to `A`.

From CLP-A:

```
SUBNAME    S STATE_TIME
---------- - --------------------------
T10001     A 2006-02-22-18.32.22.112000
T10002     A 2006-02-22-18.37.33.470001
T10003     A 2006-02-22-18.43.44.954000
```

From CLP-B:

```
SUBNAME    S STATE_TIME
---------- - --------------------------
T10004     A 2006-02-22-18.43.37.333001
T10005     A 2006-02-22-18.37.36.394000
T10006     A 2006-02-22-18.43.47.357000
```

From CLP-C:

```
SUBNAME    S STATE_TIME
---------- - --------------------------
T10007     A 2006-02-22-18.43.34.519000
T10008     A 2006-02-22-18.37.33.580000
T10009     A 2006-02-22-18.43.49.551000
```

From CLP-D:

```
SUBNAME    S STATE_TIME
---------- - --------------------------
T10010     A 2006-02-22-18.43.30.002001
T10011     A 2006-02-22-18.43.45.064000
T10012     A 2006-02-22-18.43.45.064001
```

We can see that we now have an active Q subscription group.

The order of Q subscription activation is shown in the following table. The first row shows the system name, and the row below that shows the Q subscription name. The initial row shows the Q subscriptions that are active once Q Capture and Q Apply are started. The `db2ac` row shows the Q subscriptions that are active after the `asnclp -f SYSA_qsub_start_db2ac.asnclp` command has been executed. The `db2ad` row shows the Q subscriptions that are active after the `asnclp -f SYSA_qsub_start_db2ad.asnclp` command has been executed.

P2P four-way—order of Q subscription activation is shown in the following table:

	A			**B**			**C**			**D**		
`T100`	01	02	03	04	05	06	07	08	09	10	11	12
Initial	A			A								
`db2ac`	A	A		A	A		A	A				
`db2ad`	A	A	A	A	A	A	A	A	A	A	A	A

Testing replication

We are now in a position to test our P2P four-way replication setup. We can insert a record into `ERIC.T1` on `DB2A` and check that it is replicated to `DB2B`, `DB2C`, and `DB2D`.

From CLP-A, issue:

```
$ db2 "insert into eric.t1(c1,c2,c3) values (1,1,'J')"
```

From CLP-B, issue:

```
$ db2 "select * from eric.t1"
```

We should see one record in `ERIC.T1` on `DB2B`.

From CLP-C, issue:

```
$ db2 "select * from eric.t1"
```

We should see one record in `ERIC.T1` on `DB2C`.

From CLP-D, issue:

```
$ db2 "select * from eric.t1"
```

We should see one record in `ERIC.T1` on `DB2D`.

And then if we insert a record into ERIC.T1 on DB2D and check that it is replicated to DB2A, DB2B, and DB2C:

From CLP-D, issue:

```
$ db2 "insert into eric.t1(c1,c2,c3) values (2,2,'H')"
```

From CLP-A, issue:

```
$ db2 "select * from eric.t1"
```

We should see two records in ERIC.T1 on DB2A.

From CLP-B, issue:

```
$ db2 "select * from eric.t1"
```

We should see two records in FRED.T1 on DB2B.

From CLP-C, issue:

```
$ db2 "select * from eric.t1"
```

We should see two records in FRED.T1 on DB2C.

And then if we insert a record into ERIC.T1 on DB2C and check that it is replicated to DB2D, DB2A, and DB2B:

From CLP-C, issue:

```
$ db2 "insert into eric.t1(c1,c2,c3) values (3,3,'S')"
```

From CLP-D, issue:

```
$ db2 "select * from eric.t1"
```

We should see three records in ERIC.T1 on DB2D.

From CLP-A, issue:

```
$ db2 "select * from eric.t1"
```

We should see three records in ERIC.T1 on DB2A.

From CLP-B, issue:

```
$ db2 "select * from eric.t1"
```

We should see three records in ERIC.T1 on DB2B.

And finally if we insert a record into `ERIC.T1` on `DB2B` and check that it is replicated to `DB2C`, `DB2D`, and `DB2A`:

From CLP-B, issue:

```
$ db2 "insert into eric.t1(c1,c2,c3) values (4,4,'T')"
```

From CLP-C, issue:

```
$ db2 "select * from eric.t1"
```

We should see four records in `ERIC.T1` on `DB2C`.

From CLP-D, issue:

```
$ db2 "select * from eric.t1"
```

We should see four records in `ERIC.T1` on `DB2D`.

From CLP-A, issue:

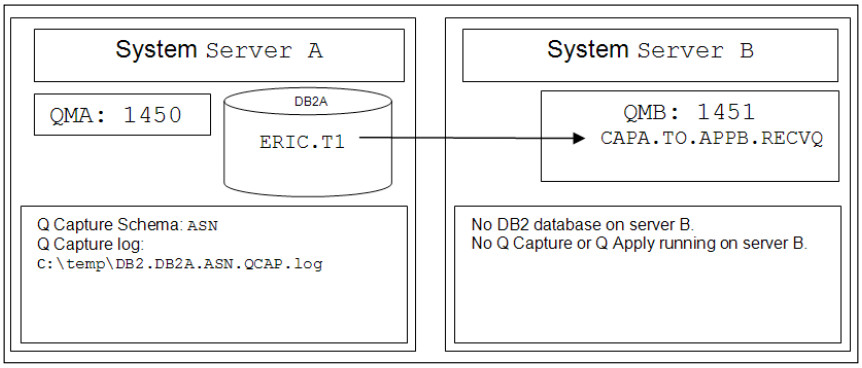
```
$ db2 "select * from eric.t1"
```

We should see four records in `ERIC.T1` on `DB2A`.

We can see that the P2P four-way Q replication setup is working.

Event Publishing

The setup we will use is shown in the following diagram:



The database layer

We need to create one source database called `DB2A`, follow the instructions in the *First steps – Database creation* section. Note that there is no target database/table.

We now have the database layer defined and can now proceed to the WebSphere MQ layer.

The WebSphere MQ layer

This section deals with the WebSphere MQ layer, which covers creating the Queue Managers and the appropriate queues, and then starting the Listeners and Channels.

Creating the Queue Managers and Queues

We need to create and start two Queue Managers called `QMA` and `QMB`—follow the instructions in the *First steps – Queue Manager processing* section.

The queues we need for Event Publishing are shown next.

The queues we need for `QMA` are in `SYSA_QMA_MQDEFS_EP_AB.TXT` file:

```
DELETE QLOCAL(CAPA.ADMINQ) PURGE

DEFINE QLOCAL(CAPA.ADMINQ) +

REPLACE +

DESCR('LOCAL DEFN OF ADMINQ FOR
CAPA CAPTURE') +

PUT(ENABLED) +

GET(ENABLED) +

SHARE +

DEFSOPT(SHARED) +

DEFPSIST(YES)


DELETE QLOCAL(CAPA.RESTARTQ)
PURGE

DEFINE QLOCAL(CAPA.RESTARTQ) +
```

```
DEFINE QREMOTE(CAPA.TO.APPB.
SENDQ.REMOTE) +

REPLACE +

DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPA TO APPB') +

PUT(ENABLED) +

XMITQ(QMB.XMITQ)  +

RNAME(CAPA.TO.APPB.RECVQ) +

RQMNAME(QMB) +

DEFPSIST(YES)


DEFINE QLOCAL(QMB.XMITQ) +

REPLACE +

DESCR('TRANSMISSION QUEUE TO
QMB') +
```

```
REPLACE +
DESCR('LOCAL DEFN OF RESTART FOR
CAPA CAPTURE') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)

DEFINE QMODEL(IBMQREP.SPILL.
MODELQ) +
REPLACE +
DEFSOPT(SHARED) +
MAXDEPTH(500000) +
MSGDLVSQ(FIFO) +
DEFTYPE(PERMDYN)

DEFINE QLOCAL(DEAD.LETTER.QUEUE.
QMA) +
REPLACE +
DESCR('LOCAL DEAD LETTER QUEUE
QMA') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)
USAGE(XMITQ) +
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
TRIGTYPE(FIRST) +
TRIGDATA(QMA.TO.QMB) +
INITQ(SYSTEM.CHANNEL.INITQ)

DEFINE CHANNEL(QMA.TO.QMB) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMB') +
XMITQ(QMB.XMITQ) +
CONNAME('127.0.0.1(1451)')

DEFINE CHANNEL(QMB.TO.QMA) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM
QMB')
```

From CLP-A, run the file as:

```
$ runmqsc QMA < SYSA_QMA_MQDEFS_EP_AB.TXT
```

The queues we need for QMB are in SYSB_QMB_MQDEFS_EP_AB.TXT file:

```
DEFINE QMODEL(IBMQREP.SPILL.
MODELQ) +
REPLACE +
DEFSOPT(SHARED) +
MAXDEPTH(500000) +
MSGDLVSQ(FIFO) +
DEFTYPE(PERMDYN)


DEFINE QLOCAL(DEAD.LETTER.QUEUE.
QMB) +
REPLACE +
DESCR('LOCAL DEAD LETTER QUEUE
QMB') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)


DEFINE QLOCAL(CAPA.TO.APPB.RECVQ)
+
REPLACE +
DESCR('LOCAL RECEIVE QUEUE - APPB
FROM CAPA') +
PUT(ENABLED) +
GET(ENABLED) +
XMITQ(QMA.XMITQ) +
RNAME(CAPA.ADMINQ) +
RQMNAME(QMA) +
DEFPSIST(YES)


DEFINE QLOCAL(QMA.XMITQ) +
REPLACE +
DESCR('TRANSMISSION QUEUE TO
QMA') +
USAGE(XMITQ) +
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
TRIGTYPE(FIRST) +
TRIGDATA(QMB.TO.QMA) +
INITQ(SYSTEM.CHANNEL.INITQ)


DEFINE CHANNEL(QMB.TO.QMA) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMA') +
XMITQ(QMA.XMITQ) +
```

```
DEFSOPT(SHARED) +
DEFPSIST(YES)

DEFINE QREMOTE(CAPA.ADMINQ.
REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF ADMINQ FOR
CAPA CAPTURE') +
PUT(ENABLED) +
```

```
CONNAME('127.0.0.1(1450)')

DEFINE CHANNEL(QMA.TO.QMB) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM
QMA')
```

From CLP-B, run the file as:

```
$ runmqsc QMB < SYSB_QMB_MQDEFS_EP_AB.TXT
```

Starting the Listeners

Start the Listeners for QMA and QMB as described in the *Bidirectional replication – The WebSphere MQ layer – Starting the Listeners* section.

Starting the Channels

Start the Channels between QMA and QMB as described in the *Bidirectional replication – The WebSphere MQ layer – Starting the Channels* section.

Testing the WebSphere MQ layer

Now that everything is started, we need to test the MQ layer.

Follow the instructions in the *Unidirectional replication – The WebSphere MQ layer – Testing the WebSphere MQ layer* section.

We have now defined the database and WebSphere MQ layers, and can proceed to the Q replication layer.

The Q replication layer

The following sections give the ASNCLP commands to create the control tables, the Publication Queue Map, and the XML Publication. The tasks are:

- Creating the Q Capture tables on `DB2A`
- Creating a Publication Queue Map from `DB2A` to `DB2B`
- Creating an XML Publication

Creating Q Capture control tables on DB2A

Follow the instructions in the *Unidirectional replication – The Q replication layer – Creating Q Capture control tables on DB2A* section.

Creating a Publication Queue Map from DB2A to DB2B

The Publication Queue Map for the queues going from `DB2A` to `DB2B` can be created using the following ASNCLP commands in the `SYSA_crt_pqma2b.asnclp` file:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2A;
SET CAPTURE SCHEMA SOURCE ASN;

SET QMANAGER QMA FOR CAPTURE SCHEMA;
SET QMANAGER QMB FOR APPLY   SCHEMA;

CREATE PUBQMAP "PQMA2B"
USING
SENDQ "CAPA.TO.APPB.SENDQ.REMOTE"
MESSAGE CONTENT TYPE T
ERROR ACTION S
HEARTBEAT INTERVAL 0
HEADER NONE;
```

From CLP-A, run the file as:

```
$ asnclp -f SYSA_crt_pqma2b.asnclp
```

Creating the XML Publication

The file to create the XML publication is called `SYSA_crt_xmlpub.asnclp`.

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;
SET SERVER CAPTURE TO DB DB2A;
```

```
SET CAPTURE SCHEMA SOURCE ASN;

SET QMANAGER QMA FOR CAPTURE SCHEMA;
SET QMANAGER QMB FOR APPLY   SCHEMA;

CREATE XML PUB  (PUBNAME PUBA2B
PUBQMAP "PQMA2B"
ERIC.T1
COLS ALL
ALL CHANGED ROWS Y
BEFORE VALUES Y
CHANGED COLS ONLY N
HAS LOAD PHASE N
SUPPRESS DELETES N);
```

From CLP-A, run the file as:

```
$ asnclp -f SYSA_crt_xmlpub.asnclp
```

Starting Q Capture on DB2A

Follow the instructions in the *Unidirectional replication – Starting Q Capture and Q Apply – Starting Q Capture on DB2A* section.

Testing publication

- To test an `INSERT` statement.

  From CLP-A, issue:

  ```
  $ db2 "insert into eric.t1 values(1,1,'J')"
  ```

  The local Receive Queue on `QMB`, which will receive the messages from `QMA` is called `CAPA.TO.APPB.RECVQ`. We cannot use the `amqsget` program to retrieve messages from this queue, because the message length is greater than 101 bytes. If we try, we will get:

  ```
  $ amqsget CAPA.TO.APPB.RECVQ QMB
  Sample AMQSGET0 start
  MQGET ended with reason code 2080
  Sample AMQSGET0 end
  ```

  The reason code `2080` indicates that the message length is greater than the 101 bytes limit.

  There are a couple of ways to view the messages on the `QMB` queue. We can use the `rfhutil` utility, or the `amqsbcg` command—we will use this command:

  ```
  $ amqsbcg CAPA.TO.APPB.RECVQ QMB
  ```

```
AMQSBCG0 - starts here
**********************
 MQOPEN - 'CAPA.TO.APPB.RECVQ'
 MQGET of message number 1
****Message descriptor****
  StrucId  : 'MD  '  Version : 2
  Report   : 0  MsgType : 8
  Expiry   : -1  Feedback : 0
  Encoding : 546  CodedCharSetId : 1208
  Format : 'MQSTR   '
  Priority : 0  Persistence : 1
  MsgId : X'515245504416E5D200000000000000000000000000000001'
  CorrelId : X'00000000000000000000000000000000000000000000000000
0'
  BackoutCount : 0
  ReplyToQ       : '
'
  ReplyToQMgr    : 'QMA
'
  ** Identity Context
  UserIdentifier : 'db2admin    '
  AccountingToken :
   X'16010515000000B4B7CD226BD6620407E53B2BEE03000000000000000000
000000B'
  ApplIdentityData : '                                '
  ** Origin Context
  PutApplType    : '11'
  PutApplName    : 's\IBM\SQLLIB\BIN\asnqcap.exe'
  PutDate  : '20060314'    PutTime  : '15483465'
  ApplOriginData : '    '
  GroupId : X'000000000000000000000000000000000000000000000000'
  MsgSeqNumber   : '1'
  Offset         : '0'
  MsgFlags       : '0'
  OriginalLength : '-1'
****   Message      ****
 length - 603 bytes
00000000:  3C3F 786D 6C20 7665 7273 696F 6E3D 2231 '<?xml
version="1'
```

```
00000010:  2E30 2220 656E 636F 6469 6E67 3D22 5554 '.0"
encoding="UT'
00000020:  462D 3822 203F 3E3C 6D73 6720 786D 6C6E 'F-8" ?><msg
xmln'
00000030:  733A 7873 693D 2268 7474 703A 2F2F 7777
's:xsi="http://ww'
00000040:  772E 7733 2E6F 7267 2F32 3030 312F 584D 'w.w3.
org/2001/XM'
00000050:  4C53 6368 656D 612D 696E 7374 616E 6365 'LSchema-
instance'
00000060:  2220 7873 693A 6E6F 4E61 6D65 7370 6163 '"
xsi:noNamespac'
00000070:  6553 6368 656D 614C 6F63 6174 696F 6E3D
'eSchemaLocation='
00000080:  226D 7163 6170 2E78 7364 2220 7665 7273 '"mqcap.xsd"
vers'
00000090:  696F 6E3D 2231 2E30 2E30 2220 6462 4E61 'ion="1.0.0"
dbNa'
000000A0:  6D65 3D22 4442 3241 223E 3C73 7562 5363
'me="DB2A"><subSc'
000000B0:  6865 6D61 2073 7562 4E61 6D65 3D22 5055 'hema
subName="PU'
000000C0:  4241 3242 2220 7372 634F 776E 6572 3D22 'BA2B"
srcOwner="'
000000D0:  4552 4943 2220 7372 634E 616D 653D 2254 'ERIC"
srcName="T'
000000E0:  3122 2073 656E 6451 4E61 6D65 3D22 4341 '1"
sendQName="CA'
000000F0:  5041 2E54 4F2E 4150 5042 2E53 454E 4451 'PA.TO.APPB.
SENDQ'
00000100:  2E52 454D 4F54 4522 2061 6C6C 4368 616E '.REMOTE"
allChan'
00000110:  6765 6452 6F77 733D 2231 2220 6265 666F 'gedRows="1"
befo'
00000120:  7265 5661 6C75 6573 3D22 3122 2063 6861
'reValues="1" cha'
00000130:  6E67 6564 436F 6C73 4F6E 6C79 3D22 3022
'ngedColsOnly="0"'
00000140:  2068 6173 4C6F 6164 5068 6173 653D 226E '
hasLoadPhase="n'
00000150:  6F6E 6522 2064 6253 6572 7665 7254 7970 'one"
dbServerTyp'
```

```
00000160:  653D 2251 4442 322F 4E54 2220 6462 496E 'e="QDB2/NT" dbIn'
00000170:  7374 616E 6365 3D22 4442 3222 2064 6252 'stance="DB2" dbR'
00000180:  656C 6561 7365 3D22 392E 312E 3022 2063 'elease="9.1.0" c'
00000190:  6170 5265 6C65 6173 653D 2239 2E31 2E30 'apRelease="9.1.0'
000001A0:  223E 3C63 6F6C 206E 616D 653D 2243 3122 '"><col name="C1"'
000001B0:  2074 7970 653D 2269 6E74 6567 6572 2220 ' type="integer" '
000001C0:  6C65 6E3D 2234 2220 636F 6465 7061 6765 'len="4" codepage'
000001D0:  3D22 3022 2069 734B 6579 3D22 3122 2F3E '="0" isKey="1"/>'
000001E0:  3C63 6F6C 206E 616D 653D 2243 3222 2074 '<col name="C2" t'
000001F0:  7970 653D 2269 6E74 6567 6572 2220 6C65 'ype="integer" le'
00000200:  6E3D 2234 2220 636F 6465 7061 6765 3D22 'n="4" codepage="'
00000210:  3022 2F3E 3C63 6F6C 206E 616D 653D 2243 '0"/><col name="C'
00000220:  3322 2074 7970 653D 2263 6861 7222 206C '3" type="char" l'
00000230:  656E 3D22 3130 2220 636F 6465 7061 6765 'en="10" codepage'
00000240:  3D22 3132 3532 222F 3E3C 2F73 7562 5363 '="1252"/></subSc'
00000250:  6865 6D61 3E3C 2F6D 7367 3E          'hema></msg>  '

 MQGET of message number 2
****Message descriptor****
  StrucId  : 'MD  '  Version : 2
  Report   : 0  MsgType : 8
  Expiry   : -1  Feedback : 0
  Encoding : 546  CodedCharSetId : 1208
  Format : 'MQSTR   '
  Priority : 0  Persistence : 1
```

```
  MsgId : X'515245504416E5D200000000000000000000000000000002'
  CorrelId : X'00000000000000000000000000000000000000000000000000
0'
  BackoutCount : 0
  ReplyToQ       : '
'
  ReplyToQMgr    : 'QMA
'
  ** Identity Context
  UserIdentifier : 'db2admin    '
  AccountingToken :
   X'16010515000000B4B7CD226BD6620407E53B2BEE03000000000000000000
000000B'
  ApplIdentityData : '                                '
  ** Origin Context
  PutApplType    : '11'
  PutApplName    : 's\IBM\SQLLIB\BIN\asnqcap.exe'
  PutDate  : '20060314'    PutTime  : '15514671'
  ApplOriginData : '    '
  GroupId : X'000000000000000000000000000000000000000000000000'
  MsgSeqNumber   : '1'
  Offset         : '0'
  MsgFlags       : '0'
  OriginalLength : '-1'
****   Message      ****
 length - 1071 bytes
00000000:  3C3F 786D 6C20 7665 7273 696F 6E3D 2231 '<?xml
version="1'
00000010:  2E30 2220 656E 636F 6469 6E67 3D22 5554 '.0"
encoding="UT'
00000020:  462D 3822 203F 3E3C 6D73 6720 786D 6C6E 'F-8" ?><msg
xmln'
00000030:  733A 7873 693D 2268 7474 703A 2F2F 7777
's:xsi="http://ww'
00000040:  772E 7733 2E6F 7267 2F32 3030 312F 584D 'w.w3.
org/2001/XM'
00000050:  4C53 6368 656D 612D 696E 7374 616E 6365 'LSchema-
instance'
00000060:  2220 7873 693A 6E6F 4E61 6D65 7370 6163 '"
xsi:noNamespac'
```

```
00000070:  6553 6368 656D 614C 6F63 6174 696F 6E3D
'eSchemaLocation='
00000080:  226D 7163 6170 2E78 7364 2220 7665 7273 '"mqcap.xsd"
vers'
00000090:  696F 6E3D 2231 2E30 2E30 2220 6462 4E61 'ion="1.0.0"
dbNa'
000000A0:  6D65 3D22 4442 3241 223E 3C74 7261 6E73
'me="DB2A"><trans'
000000B0:  2061 7574 6849 443D 2244 4232 4144 4D49 '
authID="DB2ADMI'
000000C0:  4E22 2069 734C 6173 743D 2231 2220 7365 'N"
isLast="1" se'
000000D0:  676D 656E 744E 756D 3D22 3122 2063 6D69
'gmentNum="1" cmi'
000000E0:  744C 534E 3D22 3030 3030 3A30 3030 303A
'tLSN="0000:0000:'
000000F0:  3030 3030 3A30 3138 653A 6532 3330 2220
'0000:018e:e230" '
00000100:  636D 6974 5469 6D65 3D22 3230 3036 2D30
'cmitTime="2006-0'
00000110:  332D 3134 5431 353A 3531 3A34 312E 3030 '3-
14T15:51:41.00'
00000120:  3030 3031 223E 3C69 6E73 6572 7452 6F77
'0001"><insertRow'
00000130:  2073 7562 4E61 6D65 3D22 5055 4241 3242 '
subName="PUBA2B'
00000140:  2220 2020 2020 2020 2020 2020 2020 2020 '"         '
00000150:  2020 2020 2020 2020 2020 2020 2020 2020 '          '
00000160:  2020 2020 2020 2020 2020 2020 2020 2020 '          '
00000170:  2020 2020 2020 2020 2020 2020 2020 2020 '          '
00000180:  2020 2020 2020 2020 2020 2020 2020 2020 '          '
00000190:  2020 2020 2020 2020 2020 2020 2020 2020 '          '
000001A0:  2020 2020 2020 2020 2020 2020 2020 2020 '          '
000001B0:  2020 2020 2020 2020 2020 2020 2020 2020 '          '
000001C0:  2020 2020 2020 2020 2020 2020 2020 2020 '          '
000001D0:  2020 2020 2020 2020 2020 2020 2020 2020 '          '
000001E0:  2020 2020 2020 2020 2020 2020 2020 2020 '          '
000001F0:  2020 2020 2020 2020 2020 2020 2020 2020 '          '
00000200:  2020 2020 2020 2020 2020 2020 2020 2020 '          '
00000210:  2020 2020 2020 2020 2020 2020 2020 2020 '          '
```

```
00000220:  2020 2020 2020 2020 2020 2020 2020 2020 '           '
00000230:  2020 2020 2020 2020 2020 2020 2020 2020 '           '
00000240:  2020 2020 2020 2020 2020 2020 2020 2020 '           '
00000250:  2020 2020 2020 2020 2020 2020 2020 2020 '           '
00000260:  2020 2020 2020 2020 2020 2020 2020 2020 '           '
00000270:  2020 2020 2020 2020 2020 2020 2020 2020 '           '
00000280:  2020 2020 2020 2020 2020 2020 2020 2020 '           '
00000290:  2020 2020 2020 2020 2020 2020 2020 2020 '           '
000002A0:  2020 2020 2020 2020 2020 2020 2020 2020 '           '
000002B0:  2020 2020 2020 2020 2020 2020 2020 2020 '           '
000002C0:  2020 2020 2020 2020 2020 2020 2020 2020 '           '
000002D0:  2020 2020 2020 2020 2020 2020 2020 2020 '           '
000002E0:  2020 2020 2020 2020 2020 2020 2020 2020 '           '
000002F0:  2020 2020 2020 2020 2020 2020 2020 2020 '           '
00000300:  2020 2020 2020 2020 2020 2020 2020 2020 '           '
00000310:  2020 2020 2020 2020 2020 2020 2020 2020 '           '
00000320:  2020 2020 2020 2020 2020 2020 2020 2020 '           '
00000330:  2020 2020 2020 2020 2020 2020 2020 2020 '           '
00000340:  2020 2020 2020 2020 2020 2073 7263 4F77 '     srcOw'
00000350:  6E65 723D 2245 5249 4322 2073 7263 4E61 'ner="ERIC" 
srcNa'
00000360:  6D65 3D22 5431 2220 696E 7465 6E74 5345 'me="T1" 
intentSE'
00000370:  513D 2230 3030 303A 3030 3030 3A30 3030 
'Q="0000:0000:000'
00000380:  303A 3031 3865 3A65 3161 6122 3E3C 636F 
'0:018e:e1aa"><co'
00000390:  6C20 6E61 6D65 3D22 4331 2220 6973 4B65 'l name="C1" 
isKe'
000003A0:  793D 2231 223E 3C69 6E74 6567 6572 3E31 
'y="1"><integer>1'
000003B0:  3C2F 696E 7465 6765 723E 3C2F 636F 6C3E '</
integer></col>'
000003C0:  3C63 6F6C 206E 616D 653D 2243 3222 3E3C '<col 
name="C2"><'
000003D0:  696E 7465 6765 723E 313C 2F69 6E74 6567 'integer>1</
integ'
000003E0:  6572 3E3C 2F63 6F6C 3E3C 636F 6C20 6E61 'er></
col><col na'
000003F0:  6D65 3D22 4333 223E 3C63 6861 723E 4A20 
'me="C3"><char>J '
00000400:  2020 2020 2020 2020 3C2F 6368 6172 3E3C '        </
char><'
```

```
00000410:  2F63 6F6C 3E3C 2F69 6E73 6572 7452 6F77 '/col></
insertRow'
00000420:  3E3C 2F74 7261 6E73 3E3C 2F6D 7367 3E   '></trans></
msg> '
 No more messages
 MQCLOSE
 MQDISC
C:\>
```

We can see our inserted row at the bottom of message (2). We can see the values that we entered and that it was an insert operation.

- Test an `UPDATE` statement. Update the row, then:

  From CLP-A, issue:

  ```
  $ db2 "update eric.t1 set c3 = 'T' where c1 = 1"
  ```

  Next, run the `amqsbcg` command:

  ```
  $ amqsbcg CAPA.TO.APPB.RECVQ QMB
  ```

  Only the bottom of the output is shown:

  ```
  00000340:  2020 2020 2020 2020 2020 2073 7263 4F77 '
  srcOw'
  00000350:  6E65 723D 2245 5249 4322 2073 7263 4E61 'ner="ERIC"
  srcNa'
  00000360:  6D65 3D22 5431 2220 696E 7465 6E74 5345 'me="T1"
  intentSE'
  00000370:  513D 2230 3030 303A 3030 3030 3A30 3030
  'Q="0000:0000:000'
  00000380:  303A 3031 3865 3A65 6366 6122 3E3C 636F
  '0:018e:ecfa"><co'
  00000390:  6C20 6E61 6D65 3D22 4331 2220 6973 4B65 'l name="C1"
  isKe'
  000003A0:  793D 2231 223E 3C69 6E74 6567 6572 3E3C
  'y="1"><integer><'
  000003B0:  6166 7465 7256 616C 3E31 3C2F 6166 7465
  'afterVal>1</afte'
  000003C0:  7256 616C 3E3C 2F69 6E74 6567 6572 3E3C 'rVal></
  integer><'
  000003D0:  2F63 6F6C 3E3C 636F 6C20 6E61 6D65 3D22 '/col><col
  name="'
  000003E0:  4332 223E 3C69 6E74 6567 6572 3E3C 6166
  'C2"><integer><af'
  ```

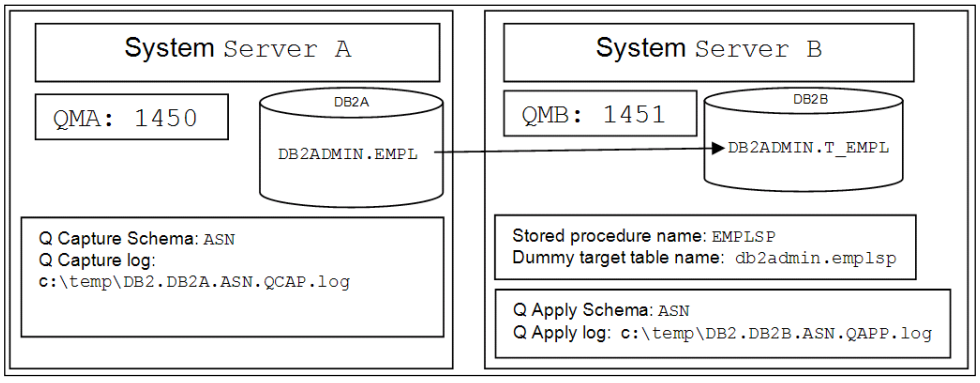
```
000003F0:  7465 7256 616C 3E31 3C2F 6166 7465 7256 'terVal>1</
afterV'
00000400:  616C 3E3C 2F69 6E74 6567 6572 3E3C 2F63 'al></
integer></c'
00000410:  6F6C 3E3C 636F 6C20 6E61 6D65 3D22 4333 'ol><col
name="C3'
00000420:  223E 3C63 6861 723E 3C62 6566 6F72 6556
'"><char><beforeV'
00000430:  616C 3E4A 2020 2020 2020 2020 203C 2F62 'al>J
</b'
00000440:  6566 6F72 6556 616C 3E3C 6166 7465 7256
'eforeVal><afterV'
00000450:  616C 3E54 2020 2020 2020 2020 203C 2F61 'al>T
</a'
00000460:  6674 6572 5661 6C3E 3C2F 6368 6172 3E3C 'fterVal></
char><'
00000470:  2F63 6F6C 3E3C 2F75 7064 6174 6552 6F77 '/col></
updateRow'
00000480:  3E3C 2F74 7261 6E73 3E3C 2F6D 7367 3E   '></trans></
msg> '
```

We can see our updated row at the bottom of the third message. We can see the values that the operation was an update operation and the before and after values of what was updated.

We can see that the Event Publishing setup is working.

Replication to a stored procedure

The setup we will use is shown in the following diagram:



The process of defining the unidirectional subscription with a stored procedure as the target is similar to defining unidirectional subscription with a target table except that in the target selection process, we specify the stored procedure as the target instead of an existing table or creating a new physical table.

The database layer

We need to create two databases—a source database called `DB2A` and a target database called `DB2B`—follow the instructions in the *First steps – Database creation* section.

We now have the database layer defined and can proceed to the WebSphere MQ layer.

The WebSphere MQ layer

This section deals with the WebSphere MQ layer, which covers creating the Queue Managers and the appropriate queues, and then starting the Listeners and Channels.

Creating the Queue Managers and Queues

We need to create and start two Queue Managers called `QMA` and `QMB`—follow the instructions in the *First steps – Queue Manager processing* section.

Create the MQ queues as for unidirectional replication, as described in the *Unidirectional replication – The WebSphere MQ layer – Creating the Queue Managers and queues* section.

Starting the Listeners

Start the Listeners for `QMA` and `QMB` as described in the *Bidirectional replication – The WebSphere MQ layer – Starting the Listeners* section.

Starting the Channels

Start the Channels between `QMA` and `QMB` as described in the *Bidirectional replication – The WebSphere MQ layer – Starting the Channels* section.

Testing the WebSphere MQ layer

Now that everything is started we need to test the MQ layer. Refer to the *Unidirectional replication – The WebSphere MQ layer – Testing the WebSphere MQ layer* section for a description of the tests.

We have now defined the database and WebSphere MQ layers, and can proceed to the Q replication layer.

The Q replication layer

The following sections give the ASNCLP commands to create the control tables, the Replication Queue Maps and the Q subscription.

- Creating the Q Capture tables on `DB2A`
- Creating the Q Apply tables on `DB2B`
- Creating the source/target tables and the stored procedure
- Creating a Replication Queue Map for `DB2A` to `DB2B`
- Creating a Q subscription

Creating Q Capture control tables on DB2A

Follow the instructions in the *Unidirectional replication – The Q replication layer – Creating Q Capture control tables on DB2A* section.

Creating Q Apply control tables on DB2B

Follow the instructions in the *Unidirectional replication – The Q replication layer – Creating Q Apply control tables on DB2B* section.

Creating the source/target tables/stored procedure

Note that the stored procedure name is the same as the source table name, refer to the *The different types of Q replication – Replicating* to a stored procedure section of *Chapter 1, Q Replication Overview*.

To create the source table `DB2ADMIN.EMPL` on `DB2A`:

From CLP-A, issue:

```
$ db2" CREATE TABLE db2admin.empl (EMPNO INTEGER NOT NULL, FIRSTNAME
VARCHAR (15) NOT NULL, LASTNAME VARCHAR (15) NOT NULL, SALARY DECIMAL
NOT NULL, PRIMARY KEY ( EMPNO))"
```

To create the actual target table `DB2ADMIN.T_EMPL` on `DB2B`:

From CLP-B, issue:

```
$ db2" create table db2admin.t_empl (op_code integer, rep_time timestamp,
empno_before integer, empno_after integer, new_firstname varchar(15),
new_lastname varchar(15), new_salary decimal)"
```

The following shows the stored procedure in a file called `STOREP.TXT`:

```
CREATE PROCEDURE DB2ADMIN.EMPLSP (
  INOUT operation         integer,
  IN    suppression_ind   VARCHAR(90) ,
  IN    SRC_COMMIT_LSN    char(10) for bit data ,
  IN    SRC_TRANS_TIME    timestamp,
  IN    XEMPNO           integer,
  IN    EMPNO             integer,
  IN    firstname           varchar(15),
  IN    lastname      varchar(15),
  IN    salary              decimal(5,0)  )

BEGIN
DECLARE    SQLCODE        INTEGER ;
DECLARE    old_empno      integer;
declare    new_empno      integer;
declare    new_firstname  varchar(15);
declare    new_lastname   varchar(15);
declare    new_salary     decimal(5,0);
DECLARE    suppressid     varchar(90);
DECLARE    operationid    integer;
declare    rep_time       timestamp;
DECLARE    EXIT HANDLER FOR SQLEXCEPTION
SET OPERATION = sqlcode;
DECLARE EXIT HANDLER FOR SQLWARNING
SET OPERATION = sqlcode;
DECLARE EXIT HANDLER FOR NOT FOUND
SET OPERATION = sqlcode;
set operationid = operation;
set rep_time =  SRC_TRANS_TIME;
set suppressid = suppression_ind;
set old_empno = xempno ;
set new_empno = empno;
set new_firstname = firstname;
set new_lastname = lastname;
set new_salary = salary;

insert into db2admin.t_empl values
(operationid, rep_time, old_empno, new_empno, new_firstname, new_
lastname,
new_salary);
set operation = sqlcode;
END   @
```

Note that the stored procedure returns an SQL return code to Q Apply (`set operation = sqlcode;`) and that we do not specify a `ROLLBACK` or `COMMIT` statement, because it is the responsibility of Q Apply to rollback or commit the transaction.

The type of operation that was performed on the source table is transmitted to the stored procedure through an operation value. The following table shows the operation values that Q Apply passes to the stored procedure and what each value means. SQL return codes that Q Apply passes to the stored pro

Op Value	Type of operation:	Op Value	Type of operation:
16	Insert	64	Delete
32	Update to non-key columns	128	Update to key columns
34	Update and append		

Create the stored procedure as follows:

From CLP-B, run the file as:

```
$ db2 -td@ -f c:\asnclp\storep.txt
```

Creating a Replication Queue Map from DB2A to DB2B

Follow the instructions in the *Unidirectional replication – The Q replication layer – Creating a Replication Queue Map from DB2A to DB2B* section.

Creating a Q subscription

The file to create a stored procedure Q subscription is called `SYSA_crt_qsub_sp.asnclp`:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2A;
SET SERVER TARGET  TO DB DB2B;
SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY   SCHEMA        ASN;
SET QMANAGER QMA FOR CAPTURE SCHEMA;
SET QMANAGER QMB FOR APPLY   SCHEMA;

CREATE QSUB
USING REPLQMAP RQMA2B
(SUBNAME EMPLT DB2ADMIN.EMPL
OPTIONS
```

```
HAS LOAD PHASE N
EXIST TARGET
NAME DB2ADMIN.EMPLSP
TYPE STOREDPROC
CONFLICT ACTION F
LOAD TYPE 0);
```

From CLP-A, issue:

```
$ asnclp -f SYSA_crt_qsub_sp.asnclp
```

Starting Q Capture and Q Apply

Now we need to start Q Capture and Q Apply.

Starting Q Capture on DB2A

Follow the instructions in the *Unidirectional replication – Starting Q Capture and Q Apply – Starting Q Capture on DB2A* section.

Starting Q Apply on DB2B

Follow the instructions in the *Unidirectional replication – Starting Q Capture and Q Apply – Starting Q Apply on DB2B* section.

Testing replication

We will test replication to a stored procedure by inserting a row into the source table `DB2ADMIN.EMPL` on `DB2A` and check the target table `DB2ADMIN.T_EMPL` on `DB2B`.

From CLP-A, issue:

```
$ db2 "insert into db2admin.empl values(12345,'Carrie','Princess',41000)"
```

Now check what has happened on `DB2B`.

From CLP-B, issue:

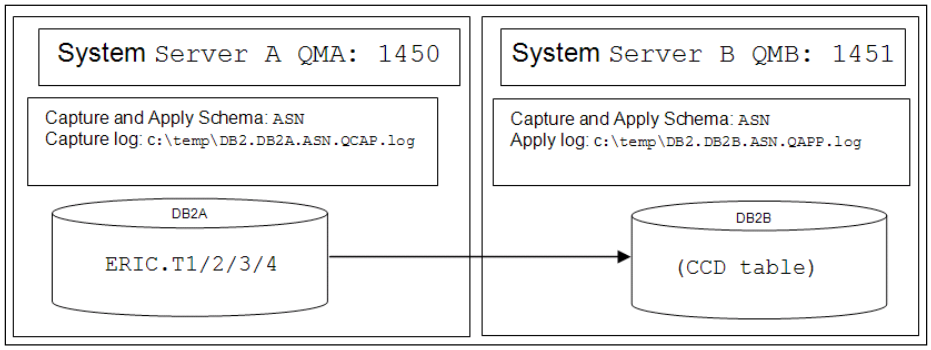
```
$ db2 "select * from db2admin.t_empl"

OP_CODE     REP_TIME                   EMPNO_BEFORE EMPNO_AFTER
         16 2006-03-15-06.30.48.000001            -       12345

NEW_FIRSTNAME   NEW_LASTNAME    NEW_SALARY
Carrie          Princess            41000.
```

We can see that the table we specified in the stored procedure (`DB2ADMIN.T_EMPL`) is populated.

We have now finished looking at replicating to a stored procedure.

Replication to a CCD table

The setup we will use is shown in the following diagram:



The test tables `ERIC.T1/2/3/4` only exist on `DB2A`—Q Apply will create the appropriate CCD tables on `DB2B`.

The database layer

We need to create two databases, a source database called `DB2A` and a target database called `DB2B`—follow the instructions in the *First steps – Database creation* section.

We now have the database layer defined and can proceed to the WebSphere MQ layer.

The WebSphere MQ layer

This section deals with the WebSphere MQ layer, which covers creating the Queue Managers and the appropriate queues, and then starting the Listeners and Channels.

Creating the Queue Managers and Queues

We need to create and start two Queue Managers called `QMA` and `QMB`—follow the instructions in the *First steps – Queue Manager processing* section.

Create the MQ queues as for unidirectional replication, described in the *Unidirectional replication – The WebSphere MQ layer – Creating the Queue Managers and queues* section.

Starting the Listeners

Start the Listeners for `QMA` and `QMB` as described in the *Bidirectional replication – The WebSphere MQ layer – Starting the Listeners* section.

Starting the Channels

Start the Channels between `QMA` and `QMB` as described in the *Bidirectional replication – The WebSphere MQ layer – Starting the Channels* section.

Testing the WebSphere MQ layer

Follow the instructions in the *Unidirectional replication – The WebSphere MQ layer – Test the WebSphere MQ layer* section.

We have now defined the database and WebSphere MQ layers, and can proceed to the Q replication layer.

The Q replication layer

The following sections give the ASNCLP commands to create the control tables, the Replication Queue Maps and the Q subscription. The tasks are:

- Creating the Q Capture tables on `DB2A`
- Creating the Q Apply tables on `DB2B`
- Creating a Replication Queue Map for `DB2A` to `DB2B`
- Creating a Q subscription

Creating Q Capture control tables on DB2A

Follow the instructions in the *Unidirectional replication – The Q replication layer – Creating Q Capture control tables on DB2A* section.

Creating Q Apply control tables on DB2B

Follow the instructions in the *Unidirectional replication – The Q replication layer – Creating Q Apply control tables on DB2B* section.

Creating a Replication Queue Map from DB2A to DB2B

Follow the instructions in the *Unidirectional replication – The Q replication layer – Creating a Replication Queue Map from DB2A to DB2B* section.

Creating a Q subscription

The files to create the CCD Q subscription are called `SYSA_crt_qsub_ccd_<type>.asnclp`. Where `<type>` is:

- `T1`: complete + condensed
- `T2`: complete + non-condensed
- `T3`: non-complete + condensed
- `T4`: non-complete + non-condensed

The `OPTIONS` block of the Q subscription definition depends on the type of CCD table required:

	COMPLETE=Y Target load options - all valid	COMPLETE=N Target load options – no load
CONDENSED=Y Primary key required	CONFLICT_ACTION=F or I CONFLICT_RULE=K OPTIONS HAS LOAD PHASE I TARGET NAME ERIC.T1 KEYS (ID) TYPE CCD COMPLETE ON CONDENSED ON KEYS (C1) CONFLICT ACTION F LOAD TYPE 0) 1	CONFLICT_ACTION=F CONFLICT_RULE=K OPTIONS HAS LOAD PHASE I TARGET NAME ERIC.T3 KEYS (ID) TYPE CCD COMPLETE OFF CONDENSED ON KEYS (C1) CONFLICT ACTION F LOAD TYPE 0) 3
CONDENSED=N No Primary key required	CONFLICT_ACTION=F CONFLICT_RULE=K OPTIONS HAS LOAD PHASE I TARGET NAME ERIC.T2 TYPE CCD COMPLETE ON CONDENSED OFF KEYS (C1) CONFLICT ACTION F LOAD TYPE 0) 2	CONFLICT_ACTION=F CONFLICT_RULE=K OPTIONS HAS LOAD PHASE I TARGET NAME ERIC.T4 TYPE CCD COMPLETE OFF CONDENSED OFF KEYS (C1) CONFLICT ACTION F LOAD TYPE 0) 4

Note that the source and target tables are different for each scenario.

T1—Q Sub for COMPLETE=Y and CONDENSED=Y

We will use a source and target table called `ERIC.T1`. The source table was created as part of the setup script, the target table will be created by Q Apply.

Create a Q subscription called `TAB1` using the following ASNCLP commands in the `SYSA_crt_qsub_ccd_t1.asnclp` file:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2A;
SET SERVER TARGET  TO DB DB2B;
SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY   SCHEMA        ASN;

SET QMANAGER QMA FOR CAPTURE SCHEMA;
SET QMANAGER QMB FOR APPLY   SCHEMA;

CREATE QSUB
USING REPLQMAP RQMA2B
(SUBNAME TAB1 ERIC.T1

OPTIONS
HAS LOAD PHASE I
SPILL_MODELQ "IBMQREP.SPILL.MODELQ"
TARGET NAME ERIC.T1
TYPE CCD
CONDENSED ON
COMPLETE ON
KEYS (C1)
CONFLICT RULE K
CONFLICT ACTION F
ERROR ACTION Q
LOAD TYPE 0);
```

From CLP-A, issue:

```
$ asnclp -f SYSA_crt_qsub_ccd_t1.asnclp
```

From CLP-B, issue:

```
$ db2 "select substr(subname,1,10) as subname, conflict_rule as CR,
conflict_action as CA, ccd_condensed as CON, CCD_COMPLETE as COM from
asn.ibmqrep_targets"

SUBNAME    CR CA CON COM
---------- -- -- --- ---
TAB1       K  F  Y   Y
```

For a `subname` of `TAB1`, check that the `CON` column (condensed) is `Y` and the `COM` column (complete) is `Y`.

We can check whether we are sending before values and only for columns which have been changed, using the following query:

From CLP-A, issue:

```
$ db2 "select substr(subname,1,10) as subname, before_values, changed_
cols_only from asn.ibmqrep_subs"

SUBNAME    BEFORE_VALUES CHANGED_COLS_ONLY
---------- ------------- -----------------
TAB1       Y             N
```

We can check the key column(s) of the target table using the following query:

From CLP-B, issue:

```
$ db2 "select substr(subname,1,10) as subname, is_key, substr(source_
colname,1,20) as source_column from asn.ibmqrep_trg_cols where subname =
'TAB1'"

SUBNAME    IS_KEY SOURCE_COLUMN
---------- ------ --------------------
TAB1       Y      C1
TAB1       N      C2
TAB1       N      C3
TAB1       N      IBMSNAP_COMMITSEQ
TAB1       N      IBMSNAP_INTENTSEQ
TAB1       N      IBMSNAP_LOGMARKER
TAB1       N      IBMSNAP_OPERATION
```

We should see that for our Q subscription of `TAB1`, we have `C1` as a key column.

The `IS_KEY` column contains a flag that indicates whether the source column is part of the key for the source table. If the value of this flag does not match the target table key definition, Q Apply rejects the schema message and invalidates the Q subscription:

- `Y`: The column is part of the source table key.
- `N`: The column is not part of the source table key.

Note that if we had defined the source table without a primary key or unique index and the `KEYS` keyword was not used to specify a key for replication, then the command automatically defines a replication key that includes all the replicated target columns.

So what would have happened if we had not put the in the `KEYS` parameter in our Q subscription definition, as shown next:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2A;
SET SERVER TARGET  TO DB DB2B;
SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY   SCHEMA        ASN;

SET QMANAGER QMA FOR CAPTURE SCHEMA;
SET QMANAGER QMB FOR APPLY   SCHEMA;

CREATE QSUB
USING REPLQMAP RQMA2B
(SUBNAME TAB1NK ERIC.T1
OPTIONS
HAS LOAD PHASE I
SPILL_MODELQ "IBMQREP.SPILL.MODELQ"
TARGET NAME ERIC.T1NKEYS
TYPE CCD
CONDENSED ON
COMPLETE ON
CONFLICT ACTION F
LOAD TYPE 0);
```

Then Q Apply would automatically take the key column from the source table. If we select from the `IBMQREP_TRG_COLS` table as previously (but with `subname = 'TAB1NK'`), then we see that the `IS_KEY` is still set to `Y` for column `C1`.

```
SUBNAME    IS_KEY SOURCE_COLUMN
---------- ------ --------------------
TAB1NK     Y      C1
TAB1NK     N      C2
TAB1NK     N      C3
TAB1NK     N      IBMSNAP_COMMITSEQ
TAB1NK     N      IBMSNAP_INTENTSEQ
TAB1NK     N      IBMSNAP_LOGMARKER
TAB1NK     N      IBMSNAP_OPERATION
```

So let's create a new source table called `ERIC.T1NK2`, which does not have a primary key and create a new Q subscription and see what is in the `IS_KEY` column.

From CLP-A, issue:

```
$ db2 "CREATE TABLE ERIC.T1NK2(c1 INT, c2 INT, c3 CHAR(10))"
```

If we create the Q subscription using the following commands:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2A;
SET SERVER TARGET  TO DB DB2B;
SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY   SCHEMA        ASN;
SET QMANAGER QMA FOR CAPTURE SCHEMA;
SET QMANAGER QMB FOR APPLY   SCHEMA;

CREATE QSUB
USING REPLQMAP RQMA2B
(SUBNAME TAB1NK2 ERIC.T1NK2
OPTIONS
HAS LOAD PHASE I SPILL_MODELQ "IBMQREP.SPILL.MODELQ"
TARGET NAME ERIC.T1NK2
TYPE CCD
CONDENSED ON
COMPLETE ON
CONFLICT ACTION F
LOAD TYPE 0);
```

If we select from the `IBMQREP_TRG_COLS` table as previously (but with `subname = 'TAB1NK2'`), then we see that the `IS_KEY` is set to `Y` for columns `C1`, `C2` and `C3`:

```
SUBNAME    IS_KEY SOURCE_COLUMN
---------- ------ --------------------
TAB1NK2    Y      C1
TAB1NK2    Y      C2
TAB1NK2    Y      C3
TAB1NK2    N      IBMSNAP_COMMITSEQ
TAB1NK2    N      IBMSNAP_INTENTSEQ
TAB1NK2    N      IBMSNAP_LOGMARKER
TAB1NK2    N      IBMSNAP_OPERATION
```

We can see that all the columns are keys.

Now let's go back to the original Q subscription and test replicating to this type of CCD table.

We will initially have `ERIC.T1` populated with two rows and then insert a row, update a row, and delete a row.

1. Initially populate `ERIC.T1`. From CLP-A, issue:

   ```
   $ db2 "insert into eric.t1 values (1,1,'J')"
   $ db2 "insert into eric.t1 values (2,2,'H')"
   ```

2. Start Q Capture. From CLP-A, issue:

   ```
   $ start asnqcap CAPTURE_SERVER=db2a STARTMODE=warmsi
   ```

3. Start Q Apply. From CLP-B, issue:

   ```
   $ start asnqapp APPLY_SERVER=db2b
   ```

4. Check if the initial load has been performed. From CLP-B, issue:

   ```
   $ db2 "select * from eric.t1"
   C1          C2          C3         IBMSNAP_INTENTSEQ        . .
   .>
            1          1 J          x'00000000000000000000'  . .
   .>
            2          2 H          x'00000000000000000000'
   IBMSNAP_COMMITSEQ               IBMSNAP_OPERATION IBMSNAP_
   LOGMARKER
   x'00000000000000000001'         I                 2006-06-12-
   05.26.54.781002
   x'00000000000000000001'         I                 2006-06-12-
   05.26.54.781002
   ```

   We should see two rows as shown in the preceding table. The initial load has been performed.

5. Insert a row into `ERIC.T1`. From CLP-A, issue:

   ```
   $ db2 "insert into eric.t1 values (3,3,'T')"
   ```

   Check the result on session B. From CLP-B, issue:

   ```
   $ db2 "select * from eric.t1"
   C1       C2        C3        IBMSNAP_INTENTSEQ          . . .>
            1          1 J      x'00000000000000000000'   . . . .>
            2          2 H       x'00000000000000000000'  . . . .>
            3         3 T       x'00000000000001BBADB9'   . . . .>
   ```

```
IBMSNAP_COMMITSEQ                IBMSNAP_OPERATION IBMSNAP_
LOGMARKER
x'00000000000000000001'          I                 2006-06-12-
05.26.54.781002
x'00000000000000000001'          I                 2006-06-12-
05.26.54.781002
x'00000000000001BBAE3F'          I                 2006-06-12-
04.29.14.000001
```

We can see that, we now have three rows in the table.

6. Insert another row into `ERIC.T1`. From CLP-A, issue:

```
$ db2 "insert into eric.t1 values (4,4,'T')"
```

Check the result on session B. From CLP-B, issue:

```
$ db2 "select * from eric.t1"

C1       C2          C3          IBMSNAP_INTENTSEQ
         1           1 J         x'00000000000000000000' . . . .>
         2           2 H         x'00000000000000000000' . . . .>
         3           3 T         x'00000000000001BBADB9' . . . .>
         4           4 T         x'00000000000001BBAE65' . . . .>

IBMSNAP_COMMITSEQ                 IBMSNAP_OPERATION IBMSNAP_
LOGMARKER
x'00000000000000000001'           I                 2006-06-12-
05.26.54.781002
x'00000000000000000001'           I                 2006-06-12-
05.26.54.781002
x'00000000000001BBAE3F'           I                 2006-06-12-
04.29.14.000001
x'00000000000001BBAEEB'           I                 2006-06-12-
04.30.36.000001
```

We can see that, we now have four rows in the table.

7. Update a row in `ERIC.T1`. From CLP-A, issue:

```
$ db2 "update eric.t1 set c2 = 4 where c1 = 2 "
```

Check the result on session B. From CLP-B, issue:

```
$ db2 "select * from eric.t1"

C1       C2           C3          IBMSNAP_INTENTSEQ       . . . .>
         1            1 J         x'00000000000000000000' . . . .>
         2            4 H         x'00000000000001BBAF11' . . . .>
```

```
        3           3 T        x'00000000000001BBADB9' . . . .>
        4           4 T        x'00000000000001BBAE65' . . . .>

IBMSNAP_COMMITSEQ           IBMSNAP_OPERATION IBMSNAP_LOGMARKER
x'00000000000000000001'     I                 2006-06-12-
05.26.54.781002
x'00000000000001BBAF85'     U                 2006-06-12-
04.31.23.000001
x'00000000000001BBAE3F'     I                 2006-06-12-
04.29.14.000001
x'00000000000001BBAEEB'     I                 2006-06-12-
04.30.36.000001
```

We can see the entry for the row we updated (in bold). Because we specified `COMPLETE=Y` and `CONDENSED=Y` we still only have four rows, but they represent the latest values.

8. Update the same row in `ERIC.T1`. From CLP-A, issue:

```
$ db2 "update eric.t1 set c2 = 5 where c1 = 2"
```

Check the result on session B. From CLP-B, issue:

```
$ db2 "select * from eric.t1"
```

```
C1      C2        C3        IBMSNAP_INTENTSEQ          . . . .>
        1           1 J       x'00000000000000000000'  . . . .>
        2           5 H       x'00000000000001BBB272'  . . . .>
        3           3 T       x'00000000000001BBADB9'  . . . .>
        4           4 T       x'00000000000001BBAE65'  . . . .>

IBMSNAP_COMMITSEQ           IBMSNAP_OPERATION IBMSNAP_LOGMARKER
x'00000000000000000001'     I                 2006-06-12-
05.26.54.781002
x'00000000000001BBB2E6'     U                 2006-06-12-
04.34.02.000001
x'00000000000001BBAE3F'     I                 2006-06-12-
04.29.14.000001
x'00000000000001BBAEEB'     I                 2006-06-12-
04.30.36.000001
```

We can see the entry for the row we updated. Because we specified `COMPLETE=Y` and `CONDENSED=Y` we still only have four rows, but they represent the latest values.

9. Delete a row from `ERIC.T1`. From CLP-A, issue:

```
$ db2 "delete from eric.t1 where c1 = 2"
```

Check the result on session B. From CLP-B, issue:

```
$ db2 "select * from eric.t1"

C1          C2           C3      IBMSNAP_INTENTSEQ       . . . .>
        1           1 J       x'00000000000000000000'  . . . .>
        2           5 H       x'00000000000001BBB30C'  . . . .>
        3           3 T       x'00000000000001BBADB9'  . . . .>
        4           4 T       x'00000000000001BBAE65'  . . . .>

IBMSNAP_COMMITSEQ          IBMSNAP_OPERATION IBMSNAP_LOGMARKER
x'00000000000000000001'    I                 2006-06-12-
05.26.54.781002
x'00000000000001BBB391'    D                 2006-06-12-
04.35.37.000001
x'00000000000001BBAE3F'    I                 2006-06-12-
04.29.14.000001
x'00000000000001BBAEEB'    I                 2006-06-12-
04.30.36.000001
```

We can see the delete operation. We still have four rows in our target table.

We have now finished testing the CCD replication specifying `COMPLETE=Y` and `CONDENSED=Y`.

T2—Q Sub for COMPLETE=Y and CONDENSED=N

We will use a source and target table called `ERIC.T2`. Create a Q subscription called `TAB2` using the following ASNCLP `SYSA_crt_qsub_ccd_t2.asnclp` file:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2A;
SET SERVER TARGET  TO DB DB2B;
SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY   SCHEMA        ASN;

SET QMANAGER QMA FOR CAPTURE SCHEMA;
SET QMANAGER QMB FOR APPLY   SCHEMA;

CREATE QSUB
USING REPLQMAP RQMA2B
(SUBNAME TAB2 ERIC.T2
OPTIONS
```

```
HAS LOAD PHASE I
SPILL_MODELQ "IBMQREP.SPILL.MODELQ"
TARGET NAME ERIC.T2
TYPE CCD
CONDENSED OFF
COMPLETE ON
KEYS (C1)
CONFLICT ACTION F
LOAD TYPE 0);
```

From CLP-A, issue:

```
$ asnclp -f SYSA_crt_qsub_ccd_t2.asnclp
```

From CLP-B, issue:

```
$ db2 "select substr(subname,1,10) as subname, conflict_rule as CR,
conflict_action as CA, ccd_condensed as CON, CCD_COMPLETE as COM from
asn.ibmqrep_targets"
SUBNAME    CR CA CON COM
---------- -- -- --- ---
TAB1       K  F  Y   Y
TAB2       K  F  N   Y
```

For a `subname` of `TAB2` check that the `CON` column (condensed) is `N` and the `COM` column (complete) is `Y`.

From CLP-A, issue:

```
$ db2 "select substr(subname,1,10) as subname, before_values, changed_
cols_only from asn.ibmqrep_subs"
SUBNAME    BEFORE_VALUES CHANGED_COLS_ONLY
---------- ------------- -----------------
TAB1       Y             N
TAB2       Y             N
```

We should see for `TAB2` a `BEFORE_VALUES` value of `Y` and a `CHANGED_COLS_ONLY` value of `N`.

From CLP-B, issue:

```
$ db2 "select substr(subname,1,10) as subname, is_key, substr(source_
colname,1,20) as source_column from asn.ibmqrep_trg_cols where subname =
'TAB2' "
```

```
SUBNAME    IS_KEY SOURCE_COLUMN
---------- ------ --------------------
TAB2       Y      C1
TAB2       Y      C2
TAB2       Y      C3
TAB2       N      IBMSNAP_COMMITSEQ
TAB2       N      IBMSNAP_INTENTSEQ
TAB2       N      IBMSNAP_LOGMARKER
TAB2       N      IBMSNAP_OPERATION
```

Test CCD replication for `COMPLETE=Y` and `CONDENSED=N`.

We will initially have `ERIC.T2` populated by two rows and then insert a row, update a row, and delete a row.

1. Initially populate `ERIC.T2`. From CLP-A, issue:

   ```
   $ db2 "insert into eric.t2 values (1,1,'J') "
   $ db2 "insert into eric.t2 values (2,2,'H') "
   ```

2. Issue a `CAPSTART` command.

   We need to issue a `CAPSTART` command to activate the new Q subscription because Q Capture and Q Apply are already running (from the previous section).

   Use the ASNCLP `SYSA_qsub_start_t2.asnclp` file:

   ```
   ASNCLP SESSION SET TO Q REPLICATION;
   SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

   SET SERVER CAPTURE TO DB DB2A;
   SET SERVER TARGET TO DB DB2B;

   SET CAPTURE SCHEMA SOURCE ASN;
   SET APPLY SCHEMA ASN;

   START QSUB SUBNAME TAB2;
   ```

   From CLP-A, issue:

   ```
   $ asnclp -f SYSA_qsub_start_t2.asnclp
   ```

3. Check that the Q subscription is active. From CLP-A, issue:

   ```
   $ db2 "select substr(subname,1,10) as subname, state from asn.
   ibmqrep_subs"

   SUBNAME    STATE
   ---------- -----
   TAB1       A
   TAB2       A
   ```

We can see that our Q subscription for `TAB2` has a `STATE` of `A` which means active.

4. Check if the initial load has been performed. From CLP-B, issue:

```
$ db2 "select * from eric.t2"

C1          C2        C3        IBMSNAP_INTENTSEQ      . . . . .>
       1           1 J         x'00000000000000000000' . . . . .>
         2          2 H        x'00000000000000000000' . . . . .>

IBMSNAP_COMMITSEQ           IBMSNAP_OPERATION IBMSNAP_LOGMARKER
x'00000000000000000001'     I                 2006-06-12-
05.45.05.859004
x'00000000000000000001'     I                 2006-06-12-
05.45.05.859004
```

We can see that we have two records in our target table. The initial load has been performed.

5. Insert a row into `ERIC.T2`. From CLP-A, issue:

```
$ db2 "insert into eric.t2 values (3,3,'T') "
```

Check the result on session B. From CLP-B, issue:

```
$ db2 "select * from eric.t2"

C1           C2           C3          IBMSNAP_INTENTSEQ
1            1 J           x'00000000000000000000' . . . . .>
           2            2 H           x'00000000000000000000' . .
. . .>
           3            3 T           x'00000000000001BBD70C' . .
. . .>


IBMSNAP_COMMITSEQ               IBMSNAP_OPERATION IBMSNAP_
LOGMARKER
x'00000000000000000001'         I                 2006-06-12-
05.45.05.859004
x'00000000000000000001'         I                 2006-06-12-
05.45.05.859004
x'00000000000001BBD792'         I                 2006-06-12-
04.46.15.000001
```

We can see that, we now have three rows in the table.

6. Insert another row into ERIC.T2. From CLP-A, issue:

```
$ db2 "insert into eric.t2 values (4,4,'T') "
```

Check the result on session B. From CLP-B, issue:

```
$ db2 "select * from eric.t2"
C1          C2          C3         IBMSNAP_INTENTSEQ    . . . . .>
          1           1 J     x'00000000000000000000' . . . . .>
          2           2 H     x'00000000000000000000' . . . . .>
          3           3 T     x'00000000000001BBD70C' . . . . .>
          4           4 T     x'00000000000001BBD7B8' . . . . .>

IBMSNAP_COMMITSEQ          IBMSNAP_OPERATION IBMSNAP_LOGMARKER
x'00000000000000000001'    I                 2006-06-12-
05.45.05.859004
x'00000000000000000001'    I                 2006-06-12-
05.45.05.859004
x'00000000000001BBD792'    I                 2006-06-12-
04.46.15.000001
x'00000000000001BBD83E'    I                 2006-06-12-
04.47.25.000001
```

We can see that we now have four rows in the table.

7. Update a row in ERIC.T2. From CLP-A, issue:

```
$ db2 "update eric.t2 set c2 = 4 where c1 = 2"
```

Check the result on session B. From CLP-B, issue:

```
$ db2 "select * from eric.t2"
C1          C2          C3        IBMSNAP_INTENTSEQ     . . . . .>
          1           1 J     x'00000000000000000000' . . . . .>
          2           2 H     x'00000000000000000000'  . . . . .
          3           3 T     x'00000000000001BBD70C' . . . . .>
          4           4 T     x'00000000000001BBD7B8' . . . . .>
          2           4 H     x'00000000000001BBDC01' . . . . .>

IBMSNAP_COMMITSEQ          IBMSNAP_OPERATION IBMSNAP_LOGMARKER
x'00000000000000000001'    I                 2006-06-12-
05.45.05.859004
x'00000000000000000001'    I                 2006-06-12-
05.45.05.859004
```

```
x'00000000000001BBD792'    I                2006-06-12-
04.46.15.000001
x'00000000000001BBD83E'    I                2006-06-12-
04.47.25.000001
x'00000000000001BBDC75'    U                2006-06-12-
04.48.40.000001
```

We can see that we now have five rows in our target table. Because we specified `COMPLETE=Y` and `CONDENSED=N`.

8. Update the same row in `ERIC.T2`. From CLP-A, issue:

```
$ db2 "update eric.t2 set c2 = 5 where c1 = 2"
```

Check the result on session B. From CLP-B, issue:

```
$ db2 "select * from eric.t2"
```

```
C1          C2          C3       IBMSNAP_INTENTSEQ     . . . . .>
          1           1 J     x'00000000000000000000' . . . . .>
          2           2 H     x'00000000000000000000' . . . . .>
          3           3 T     x'00000000000001BBD70C' . . . . .>
          4           4 T     x'00000000000001BBD7B8' . . . . .>
          2           4 H     x'00000000000001BBDC01' . . . . .>
          2           5 H     x'00000000000001BBDC9B' . . . . .>

IBMSNAP_COMMITSEQ          IBMSNAP_OPERATION IBMSNAP_LOGMARKER
x'00000000000000000001'    I                2006-06-12-
05.45.05.859004
x'00000000000000000001'    I                2006-06-12-
05.45.05.859004
x'00000000000001BBD792'    I                2006-06-12-
04.46.15.000001
x'00000000000001BBD83E'    I                2006-06-12-
04.47.25.000001
x'00000000000001BBDC75'    U                2006-06-12-
04.48.40.000001
x'00000000000001BBDD0F'    U                2006-06-12-
04.49.56.000001
```

We can see that we now have six rows in our target table.

9. Delete a row from `ERIC.T2`. From CLP-A, issue:

```
$ db2 "delete from eric.t2 where c1 = 2"
```

Check the result on session B. From CLP-B, issue:

```
$ db2 "select * from eric.t2"

C1          C2          C3     IBMSNAP_INTENTSEQ       . . . . .>
          1           1 J     x'00000000000000000000' . . . . .>
          2           2 H     x'00000000000000000000' . . . . .>
          3           3 T     x'00000000000001BBD70C' . . . . .>
          4           4 T     x'00000000000001BBD7B8' . . . . .>
          2           4 H     x'00000000000001BBDC01' . . . . .>
          2           5 H     x'00000000000001BBDC9B' . . . . .>
          2           5 H     x'00000000000001BBDD35' . . . . .>

IBMSNAP_COMMITSEQ          IBMSNAP_OPERATION IBMSNAP_LOGMARKER
x'00000000000000000001'    I                 2006-06-12-
05.45.05.859004
x'00000000000000000001'    I                 2006-06-12-
05.45.05.859004
x'00000000000001BBD792'    I                 2006-06-12-
04.46.15.000001
x'00000000000001BBD83E'    I                 2006-06-12-
04.47.25.000001
x'00000000000001BBDC75'    U                 2006-06-12-
04.48.40.000001
x'00000000000001BBDD0F'    U                 2006-06-12-
04.49.56.000001
x'00000000000001BBDDBA'    D                 2006-06-12-
04.51.08.000001
```

We now have seven rows in our target table. We can see the delete operation.

We have now finished testing the CCD replication specifying `COMPLETE=Y` and `CONDENSED=N`.

T3—Q Sub for COMPLETE=N and CONDENSED=Y

We will use a source and target table called `ERIC.T3`.

Remember from the matrix, if we specify *non complete*, then the option for `HAS LOAD PHASE` must be `N` (if we specify `I`, we will get an `ASN2404E` error message when we try and create the Q subscription).

Create a Q subscription called `TAB3` using the ASNCLP `SYSA_crt_qsub_ccd_t3.asnclp` file:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2A;
SET SERVER TARGET  TO DB DB2B;
SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY   SCHEMA        ASN;
SET QMANAGER QMA FOR CAPTURE SCHEMA;
SET QMANAGER QMB FOR APPLY   SCHEMA;

CREATE QSUB
USING REPLQMAP RQMA2B
(SUBNAME TAB3 ERIC.T3
OPTIONS
HAS LOAD PHASE N
TARGET NAME ERIC.T3
TYPE CCD
CONDENSED ON
COMPLETE OFF
KEYS(C1)
CONFLICT ACTION F
LOAD TYPE 0);
```

From CLP-A, issue:

```
$ asnclp -f SYSA_crt_qsub_ccd_t3.asnclp

$ db2 "select substr(subname,1,10) as subname, before_values, changed_
cols_only from asn.ibmqrep_subs"

SUBNAME    BEFORE_VALUES CHANGED_COLS_ONLY
---------- ------------- -----------------
TAB1       Y             N
TAB2       Y             N
TAB3       Y             N
```

We should see for `TAB3` a `BEFORE_VALUES` value of `Y` and a `CHANGED_COLS_ONLY` value of `N`.

From CLP-B, issue:

```
$ db2 "select substr(subname,1,10) as subname, conflict_rule as CR,
conflict_action as CA, ccd_condensed as CON, CCD_COMPLETE as COM from
asn.ibmqrep_targets"
```

```
SUBNAME    CR CA CON COM
---------- -- -- --- ---
TAB1       K  F  Y   Y
TAB2       K  F  N   Y
TAB3       K  F  Y   N
```

For a `subname` of `TAB3` check that the `CON` column (condensed) is `Y` and the `COM` column (complete) is `N`.

From CLP-B, issue:

```
$ db2 "select substr(subname,1,10) as subname, is_key, substr(source_
colname,1,20) as source_column from asn.ibmqrep_trg_cols where subname =
'TAB3' "

SUBNAME    IS_KEY SOURCE_COLUMN
---------- ------ --------------------
TAB3       Y      C1
TAB3       N      C2
TAB3       N      C3
TAB3       N      IBMSNAP_COMMITSEQ
TAB3       N      IBMSNAP_INTENTSEQ
TAB3       N      IBMSNAP_LOGMARKER
TAB3       N      IBMSNAP_OPERATION
```

We can see that the key column is `c1`. Now we can test CCD replication for `COMPLETE=N` and `CONDENSED=Y`.

We will initially have `ERIC.T3` populated by two rows and then insert a row, update a row, and delete a row.

1. Initially populate `ERIC.T3`. From CLP-A, issue:

   ```
   $ db2 "insert into eric.t3 values (1,1,'J') "
   $ db2 "insert into eric.t3 values (2,2,'H') "
   ```

2. Issue a `CAPSTART` command. Use the ASNCLP `SYSA_qsub_start_t3.asnclp` file:

   ```
   ASNCLP SESSION SET TO Q REPLICATION;
   SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

   SET SERVER CAPTURE TO DB DB2A;
   SET SERVER TARGET TO DB DB2B;

   SET CAPTURE SCHEMA SOURCE ASN;
   ```

```
SET APPLY SCHEMA ASN;

START QSUB SUBNAME TAB3;
```

From CLP-A, issue:

```
$ asnclp -f SYSA_qsub_start_t3.asnclp
```

3. Check that the Q subscription is active. From CLP-A, issue:

```
$ db2 "select substr(subname,1,10) as subname, state from asn.
ibmqrep_subs"

SUBNAME    STATE
---------- -----
TAB1       A
TAB2       A
TAB3       A
```

We can see that our Q subscription for `TAB3` has a `STATE` of `A` which means active.

4. Check if an initial load has been performed. From CLP-B, issue:

```
$ db2 "select * from eric.t3"

C1          C2          C3          IBMSNAP_INTENTSEQ
IBMSNAP_COMMITSEQ                      IBMSNAP_OPERATION
IBMSNAP_LOGMARKER
----------- ----------- ---------- --------------------------
```

We can see that there are no records in the target table. An initial load has NOT been performed.

5. Insert a row into `ERIC.T3`. From CLP-A, issue:

```
$ db2 "insert into eric.t3 values (3,3,'T') "
```

Check the result on session B. From CLP-B, issue:

```
$ db2 "select * from eric.t3"

C1          C2          C3      IBMSNAP_INTENTSEQ     . . . . .>
          3           3 T     x'00000000000001BBF930' . . . . .>

IBMSNAP_COMMITSEQ          IBMSNAP_OPERATION IBMSNAP_LOGMARKER
x'00000000000001BBF9B6'    I                 2006-06-12-
04.59.24.000001
```

We can see that, we now have one row in the target table, which is our inserted row.

6. Insert another row into ERIC.T3. From CLP-A, issue:

```
$ db2 "insert into eric.t3 values (4,4,'T') "
```

Check the result on session B. From CLP-B, issue:

```
$ db2 "select * from eric.t3"

C1          C2          C3      IBMSNAP_INTENTSEQ     . . . . .>
          3          3 T      x'00000000000001BBF930' . . . . .>
          4           4 T      x'00000000000001BBF9DC' . . . . .

IBMSNAP_COMMITSEQ          IBMSNAP_OPERATION IBMSNAP_LOGMARKER
x'00000000000001BBF9B6'    I                 2006-06-12-
04.59.24.000001
x'00000000000001BBFA62'    I                 2006-06-12-
05.00.30.000001
```

We can see that, we now have two rows in the target table, which are our inserted rows.

7. Update a row in ERIC.T3. From CLP-A, issue:

```
$ db2 "update eric.t3 set c2 = 4 where c1 = 3"
```

Check the result on session B. From CLP-B, issue:

```
$ db2 "select * from eric.t3"
C1          C2         C3     IBMSNAP_INTENTSEQ       . . . . .>
          3          4 T      x'00000000000001BBFA88' . . . . .>
          4          4 T      x'00000000000001BBF9DC' . . . . .>

IBMSNAP_COMMITSEQ          IBMSNAP_OPERATION IBMSNAP_LOGMARKER
x'00000000000001BBFAFC'    U                 2006-06-12-
05.01.40.000001
x'00000000000001BBFA62'    I                 2006-06-12-
05.00.30.000001
```

We can see that we now have two rows in our target table. The highlighted letter shown is our updated row.

8. Update the same row in ERIC.T3. From CLP-A, issue:

```
$ db2 "update eric.t3 set c2 = 5 where c1 = 2"
```

Check the result on session B. From CLP-B, issue:

```
$ db2 "select * from eric.t3"
C1          C2          C3       IBMSNAP_INTENTSEQ    . . . . .>
```

```
           3         4 T      x'00000000000001BBFA88' . . . . .>
           4         4 T      x'00000000000001BBF9DC' . . . . .>
           2         5 H      x'00000000000001BBFCF3' . . . . .>

IBMSNAP_COMMITSEQ          IBMSNAP_OPERATION IBMSNAP_LOGMARKER
x'00000000000001BBFAFC'    U                 2006-06-12-
05.01.40.000001
x'00000000000001BBFA62'    I                 2006-06-12-
05.00.30.000001
x'00000000000001BBFD67'    U                 2006-06-12-
05.03.44.000001
```

We can see that we have three rows in our target table. The last row shows our second `UPDATE` command.

9. Delete a row from `ERIC.T3`. From CLP-A, issue:

```
$ db2 "delete from eric.t3 where c1 = 2"
```

Check the result on session B. From CLP-B, issue:

```
$ db2 "select * from eric.t3"
```

```
C1          C2        C3      IBMSNAP_INTENTSEQ      . . . . .>
          3          4 T      x'00000000000001BBFA88' . . . . .
          4         4 T      x'00000000000001BBF9DC' . . . . .>
          2         5 H      x'00000000000001BBFD8D' . . . . .>

IBMSNAP_COMMITSEQ          IBMSNAP_OPERATION IBMSNAP_LOGMARKER
x'00000000000001BBFAFC'    U                 2006-06-12-
05.01.40.000001
x'00000000000001BBFA62'    I                 2006-06-12-
05.00.30.000001
x'00000000000001BBFE12'    D                 2006-06-12-
05.04.59.000001
```

We can see that we still have three rows in our target table. We can see our delete operation.

We have now finished testing the CCD replication specifying `COMPLETE=N` and `CONDENSED=Y`.

T4—Q Sub for COMPLETE=N and CONDENSED=N

We will use a source and target table called `ERIC.T4`.

Create a Q subscription called `TAB4` using the ASNCLP `SYSA_crt_qsub_ccd_t4.asnclp` file:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2A;
SET SERVER TARGET  TO DB DB2B;
SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY   SCHEMA        ASN;

SET QMANAGER QMA FOR CAPTURE SCHEMA;
SET QMANAGER QMB FOR APPLY   SCHEMA;

CREATE QSUB
USING REPLQMAP RQMA2B
(SUBNAME TAB4 ERIC.T4
OPTIONS
HAS LOAD PHASE N
TARGET NAME ERIC.T4
TYPE CCD
CONDENSED OFF
COMPLETE OFF
KEYS(C1)
CONFLICT ACTION F
LOAD TYPE 0);
```

From CLP-A, issue:

```
$ asnclp -f SYSA_crt_qsub_ccd_t4.asnclp

$ db2 "select substr(subname,1,10) as subname, before_values, changed_
cols_only from asn.ibmqrep_subs "

SUBNAME    BEFORE_VALUES CHANGED_COLS_ONLY
---------- ------------- -----------------
TAB1       Y             N
TAB2       Y             N
TAB3       Y             N
TAB4       Y             N
```

We should see for `TAB4` a `BEFORE_VALUES` value of `Y` and a `CHANGED_COLS_ONLY` value of `N`.

From CLP-B, issue:

```
$ db2 "select substr(subname,1,10) as subname, conflict_rule as CR,
conflict_action as CA, ccd_condensed as CON, CCD_COMPLETE as COM from
asn.ibmqrep_targets "
SUBNAME    CR CA CON COM
---------- -- -- --- ---
TAB1       K  F  Y   Y
TAB2       K  F  N   Y
TAB3       K  F  Y   N
TAB4       K  F  N   N
```

For a `subname` of `TAB4` check that the `CON` column (condensed) is `N` and the `COM` column (complete) is `N`.

From CLP-B, issue:

```
$ db2 "select substr(subname,1,10) as subname, is_key, substr(source_
colname,1,20) as source_column from asn.ibmqrep_trg_cols where subname =
'TAB4'"
SUBNAME    IS_KEY SOURCE_COLUMN
---------- ------ --------------------
TAB4       Y      C1
TAB4       Y      C2
TAB4       Y      C3
TAB4       N      IBMSNAP_COMMITSEQ
TAB4       N      IBMSNAP_INTENTSEQ
TAB4       N      IBMSNAP_LOGMARKER
TAB4       N      IBMSNAP_OPERATION
```

Test CCD replication for `COMPLETE=N` and `CONDENSED=N`.

We will initially have `ERIC.T4` populated by two rows and then insert a row, update a row, and delete a row.

1. Initially populate `ERIC.T4`. From CLP-A, issue:

   ```
   $ db2 "insert into eric.t4 values (1,1,'J') "
   $ db2 "insert into eric.t4 values (2,2,'H') "
   ```

2. Issue a `CAPSTART` command. Use the ASNCLP `SYSA_qsub_start_t4.asnclp` file:

   ```
   ASNCLP SESSION SET TO Q REPLICATION;
   ```

```
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2A;
SET SERVER TARGET TO DB DB2B;

SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY SCHEMA ASN;

START QSUB SUBNAME TAB4;
```

From CLP-A, issue:

```
$ asnclp -f SYSA_qsub_start_t4.asnclp
```

3. Check that the Q subscription is active. From CLP-A, issue:

```
$ db2 "select substr(subname,1,10) as subname, state from asn.
ibmqrep_subs "

SUBNAME    STATE
---------- -----
TAB1       A
TAB2       A
TAB3       A
TAB4       A
```

We can see that our Q subscription for `TAB4` has a `STATE` of `A` which means active.

4. Check if an initial load has been performed. From CLP-B, issue:

```
$ db2 "select * from eric.t4 "

C1          C2          C3          IBMSNAP_INTENTSEQ
IBMSNAP_COMMITSEQ                      IBMSNAP_OPERATION
IBMSNAP_LOGMARKER
----------- ----------- ---------- --------------------------
--------------------------- ----------------- ---------------
```

We can see that, we have no records in our target table. An initial load has NOT been performed.

5. Insert a row into `ERIC.T4`. From CLP-A, issue:

```
$ db2 "insert into eric.t4 values (3,3,'T') "
```

Check the result on session B. From CLP-B, issue:

```
$ db2 "select * from eric.t4 "

C1          C2          C3       IBMSNAP_INTENTSEQ       . . . . .>
```

```
          3          3 T     x'00000000000001BC1872' . . . . .>

IBMSNAP_COMMITSEQ          IBMSNAP_OPERATION IBMSNAP_LOGMARKER
x'00000000000001BC18F8'    I                 2006-06-12-
05.17.20.000001
```

We can see that, we now have one row in the target table, which is our inserted row.

6. Insert another row into `ERIC.T4`. From CLP-A, issue:

```
$ db2 "insert into eric.t4 values (4,4,'T') "
```

Check the result on session B. From CLP-B, issue:

```
$ db2 "select * from eric.t4 "

C1          C2          C3      IBMSNAP_INTENTSEQ      . . . . .>
          3          3 T      x'00000000000001BC1872' . . . . .>
          4          4 T      x'00000000000001BC1BFB' . . . . .>

IBMSNAP_COMMITSEQ          IBMSNAP_OPERATION IBMSNAP_LOGMARKER
x'00000000000001BC18F8'    I                 2006-06-12-
05.17.20.000001
x'00000000000001BC1C81'    I                 2006-06-12-
05.18.05.000001
```

We can see that, we now have two rows in the target table, which are our two inserted rows.

7. Update a row in `ERIC.T4`. From CLP-A, issue:

```
$ db2 "update eric.t4 set c2 = 4 where c1 = 2 "
```

Check the result on session B. From CLP-B, issue:

```
$ db2 "select * from eric.t4 "

C1          C2          C3      IBMSNAP_INTENTSEQ      . . . . .>
          3          3 T      x'00000000000001BC1872' . . . . .>
          4          4 T      x'00000000000001BC1BFB' . . . . .>
          2          4 H      x'00000000000001BC1CA7' . . . . .

IBMSNAP_COMMITSEQ          IBMSNAP_OPERATION IBMSNAP_LOGMARKER
x'00000000000001BC18F8'    I                 2006-06-12-
05.17.20.000001
x'00000000000001BC1C81'    I                 2006-06-12-
05.18.05.000001
```

```
x'00000000000001BC1D1B'    U                 2006-06-12-
05.19.08.000001
```

We can see that, we now have three rows in our target table. The third row is our updated row.

8. Update the same row in `ERIC.T4`. From CLP-A, issue:

```
$ db2 "update eric.t4 set c2 = 5 where c1 = 2 "
```

Check the result on session B. From CLP-B, issue:

```
$ db2 "select * from eric.t4 "
```

```
C1          C2          C3      IBMSNAP_INTENTSEQ      . . . . .>
          3         3 T      x'00000000000001BC1872' . . . . .>
          4         4 T      x'00000000000001BC1BFB' . . . . .>
          2         4 H      x'00000000000001BC1CA7' . . . . .>
          2         5 H      x'00000000000001BC1D41' . . . . .>

IBMSNAP_COMMITSEQ          IBMSNAP_OPERATION IBMSNAP_LOGMARKER
x'00000000000001BC18F8'    I                 2006-06-12-
05.17.20.000001
x'00000000000001BC1C81'    I                 2006-06-12-
05.18.05.000001
x'00000000000001BC1D1B'    U                 2006-06-12-
05.19.08.000001
x'00000000000001BC1DB5'    U                 2006-06-12-
05.20.40.000001
```

We can see that, we now have four rows in our target table. The fourth row is our second update command.

9. Delete a row from `ERIC.T4`. From CLP-A, issue:

```
$ db2 "delete from eric.t4 where c1 = 2 "
```

Check the result on session B. From CLP-B, issue:

```
$ db2 "select * from eric.t4 "
```

```
C1          C2          C3      IBMSNAP_INTENTSEQ      . . . . .>
          3         3 T      x'00000000000001BC1872' . . . . .>
          4         4 T      x'00000000000001BC1BFB' . . . . .>
          2         4 H      x'00000000000001BC1CA7' . . . . .>
          2         5 H      x'00000000000001BC1D41' . . . . .>
          2         5 H      x'00000000000001BC1DDB' . . . . .>
```

```
IBMSNAP_COMMITSEQ          IBMSNAP_OPERATION IBMSNAP_LOGMARKER
x'00000000000001BC18F8'    I                 2006-06-12-
05.17.20.000001
x'00000000000001BC1C81'    I                 2006-06-12-
05.18.05.000001
x'00000000000001BC1D1B'    U                 2006-06-12-
05.19.08.000001
x'00000000000001BC1DB5'    U                 2006-06-12-
05.20.40.000001
x'00000000000001BC1E60'    D                 2006-06-12-
05.21.56.000001
```

We now have five rows in our target table. We can see the delete operation.

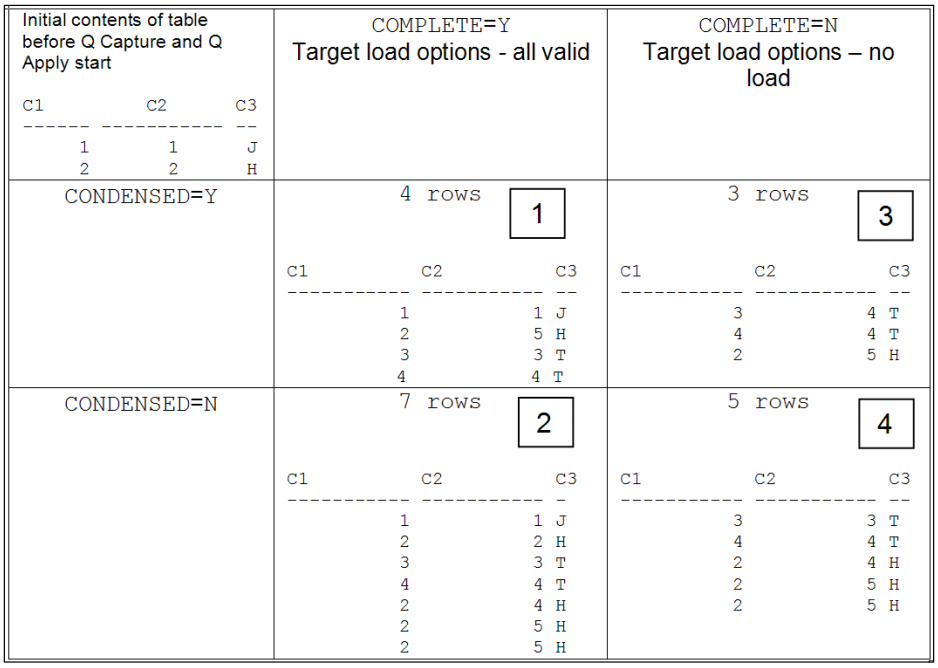
We have now finished testing the CCD replication specifying `COMPLETE=N` and `CONDENSED=N`.

Summary of CCD table options

The following table shows the order of the seven operations in the tests— two operations before and five operations after the start of Q Capture and Q Apply:

Insert row 1	`db2 insert into <table> values (1,1,'J')`
Insert row 2	`db2 insert into <table> values (2,2,'H')`
Start Q Capture and Q Apply	
Insert row 3	`db2 insert into <table> values (3,3,'T')`
Insert row 4	`db2 insert into <table> values (4,4,'T')`
Update row 2	`db2 update <table> set c2 = 4 where c1 = 2`
Update row 2	`db2 update <table> set c2 = 5 where c1 = 2`
Delete row 2	`db2 delete from <table> where c1 = 2`

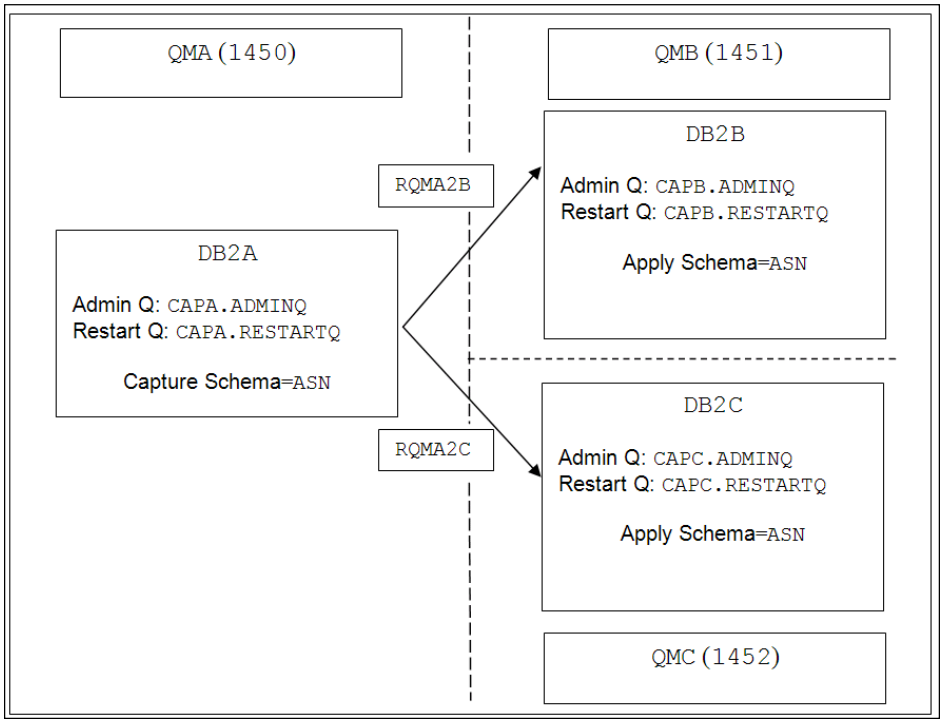
The following diagram summarizes the results from the previous tests. The order of operations was shown in the previous table:



The definitions of `COMPLETE` and `CONDENSED` were covered in the *The different types of Q replication – Replicating to a Consistent Change Data table* section of *Chapter 1, Q Replication Overview*.

Unidirectional replication to two targets (U-tree)

In this section, we look at setting up unidirectional replication from one source to many targets. Say we want to go from one source database (`DB2A`) on `SY1` to two target databases (`DB2B` and `DB2C`) on `SY2` and `SY3` respectively.



The test table `ERIC.T1` only exists on `DB2A`—Q Apply will create the test table `FRED.T1` on the other databases. `SY1` will have one Q Capture with two Send Queues, and there is a Q Apply on `SY2` and on `SY3`.

The database layer

We need to create three databases—a source database called `DB2A` and two target databases called `DB2B` and `DB2C`—follow the instructions in the *First steps – Database creation* section.

We now have defined the database layer and can proceed to the WebSphere MQ layer.

The WebSphere MQ layer

This section deals with the WebSphere MQ layer, which covers creating the Queue Managers and the appropriate queues, and then starting the Listeners and Channels.

Creating the Queue Managers and Queues

We need to create and start three Queue Managers called `QMA`, `QMB`, and `QMC`—follow the instructions in the *First steps – Queue Manager processing* section.

The following shows the queues we need for unidirectional replication to two targets. We need unidirectional queues between `QMA`/`QMB` and `QMA`/`QMC`. In the following scripts, we define some queues twice, but that will not cause a problem as the second definition request will just say that the queue already exists.

On the box containing DB2A:	On the box containing DB2B:	On the box containing DB2C:
From DB2A to DB2B: QL(CAPA.ADMINQ) QL(CAPA.RESTARTQ) QM(IBMQREP.SPILL.MODELQ) QL(DEAD.LETTER.QUEUE.QMA) QR(CAPA.TO.APPC.SENDQ.REMOTE) QL(QMB.XMITQ) CH(QMA.TO.QMB) CH(QMB.TO.QMA) From DB2A to DB2C: These are the additional queues that are needed: QR(CAPA.TO.APPC.SENDQ.REMOTE) QL(QMC.XMITQ) CH(QMA.TO.QMC) CH(QMC.TO.QMA)	QM(IBMQREP.SPILL.MODELQ) QL(DEAD.LETTER.QUEUE.QMB) QL(CAPA.TO.APPB.RECVQ) QR(CAPA.ADMINQ.REMOTE) QL(QMA.XMITQ) CH(QMB.TO.QMA) CH(QMA.TO.QMB)	QM(IBMQREP.SPILL.MODELQ) QL(DEAD.LETTER.QUEUE.QMC) QL(CAPA.TO.APPC.RECVQ) QR(CAPA.ADMINQ.REMOTE) QL(QMA.XMITQ) CH(QMC.TO.QMA) CH(QMA.TO.QMC)

The queues we need to go from `QMA` to `QMB` are in `SYSA_QMA_MQDEFS_UNI_AB.TXT` file. From CLP-A, run the file as:

```
$ runmqsc QMA < SYSA_QMA_MQDEFS_UNI_AB.TXT
```

The queues we need to go from `QMA` to `QMC` are in `SYSA_QMA_MQDEFS_UNI_AC.TXT` file:

```
DELETE QLOCAL(CAPA.ADMINQ) PURGE

DEFINE QLOCAL(CAPA.ADMINQ) +

REPLACE +

DESCR('LOCAL DEFN OF ADMINQ FOR
CAPA CAPTURE') +

PUT(ENABLED) +
```

```
DEFINE QLOCAL(CAPC.TO.APPA.
RECVQ) +

REPLACE +

DESCR('LOCAL RECEIVE QUEUE -
APPA FROM CAPC') +

PUT(ENABLED) +
```

```
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DELETE QLOCAL(CAPA.RESTARTQ)
PURGE
DEFINE QLOCAL(CAPA.RESTARTQ) +
REPLACE +
DESCR('LOCAL DEFN OF RESTART FOR
CAPA CAPTURE') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QMODEL(IBMQREP.SPILL.
MODELQ) +
REPLACE +
DEFSOPT(SHARED) +
MAXDEPTH(500000) +
MSGDLVSQ(FIFO) +
DEFTYPE(PERMDYN)
*
DEFINE QLOCAL(DEAD.LETTER.QUEUE.
QMA) +
REPLACE +
DESCR('LOCAL DEAD LETTER QUEUE
QMA') +
PUT(ENABLED) +
GET(ENABLED) +
```

```
GET(ENABLED) +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QREMOTE(CAPC.ADMINQ.
REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF ADMINQ FOR
CAPC CAPTURE') +
PUT(ENABLED) +
XMITQ(QMC.XMITQ) +
RNAME(CAPC.ADMINQ) +
RQMNAME(QMC) +
DEFPSIST(YES)
*
DEFINE QLOCAL(QMC.XMITQ) +
REPLACE +
DESCR('TRANSMISSION QUEUE TO
QMC') +
USAGE(XMITQ) +
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
TRIGTYPE(FIRST) +
TRIGDATA(QMA.TO.QMC) +
INITQ(SYSTEM.CHANNEL.INITQ)
*
DEFINE CHANNEL(QMA.TO.QMC) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
```

```
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QREMOTE(CAPA.TO.APPC.
SENDQ.REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPA TO APPC') +
PUT(ENABLED) +
XMITQ(QMC.XMITQ)  +
RNAME(CAPA.TO.APPC.RECVQ) +
RQMNAME(QMC) +
DEFPSIST(YES)  *
```

```
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMC') +
XMITQ(QMC.XMITQ) +
CONNAME('127.0.0.1(1452)')
*
DEFINE CHANNEL(QMC.TO.QMA) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM
QMC')
*
```

From CLP-A, run the file as:

```
$ runmqsc QMA < SYSA_QMA_MQDEFS_UNI_AC.TXT
```

The queues we need to go from QMB to QMA are in SYSB_QMB_MQDEFS_UNI_BA.TXT file.

From CLP-B, run the file as:

```
$ runmqsc QMB < SYSB_QMB_MQDEFS_UNI_BA.TXT
```

The queues we need to go from QMC to QMA are in SYSB_QMB_MQDEFS_UNI_CA.TXT file:

```
DELETE QLOCAL(CAPC.ADMINQ) PURGE
DEFINE QLOCAL(CAPC.ADMINQ) +
REPLACE +
DESCR('LOCAL DEFN OF ADMINQ FOR
CAPC CAPTURE') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
```

```
DEFINE QLOCAL(CAPA.TO.APPC.
RECVQ) +
REPLACE +
DESCR('LOCAL RECEIVE QUEUE -
APPC FROM CAPA') +
PUT(ENABLED) +
GET(ENABLED) +
DEFSOPT(SHARED) +
```

```
DEFSOPT(SHARED) +
DEFPSIST(YES)

DELETE QLOCAL(CAPC.RESTARTQ)
PURGE
DEFINE QLOCAL(CAPC.RESTARTQ) +
REPLACE +
DESCR('LOCAL DEFN OF RESTART FOR
CAPC CAPTURE') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)

DEFINE QMODEL(IBMQREP.SPILL.
MODELQ) +
REPLACE +
DEFSOPT(SHARED) +
MAXDEPTH(500000) +
MSGDLVSQ(FIFO) +
DEFTYPE(PERMDYN)

DEFINE QLOCAL(DEAD.LETTER.QUEUE.
QMC) +
REPLACE +
DESCR('LOCAL DEAD LETTER QUEUE
QMC') +
PUT(ENABLED) +
GET(ENABLED) +
DEFPSIST(YES)

DEFINE QREMOTE(CAPA.ADMINQ.
REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF ADMINQ FOR
CAPA CAPTURE') +
PUT(ENABLED) +
XMITQ(QMA.XMITQ) +
RNAME(CAPA.ADMINQ) +
RQMNAME(QMA) +
DEFPSIST(YES)

DEFINE QLOCAL(QMA.XMITQ) +
REPLACE +
DESCR('TRANSMISSION QUEUE TO
QMA') +
USAGE(XMITQ) +
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
TRIGTYPE(FIRST) +
TRIGDATA(QMC.TO.QMA) +
INITQ(SYSTEM.CHANNEL.INITQ)

DEFINE CHANNEL(QMC.TO.QMA) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
```

```
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)

DEFINE QREMOTE(CAPC.TO.APPA.
SENDQ.REMOTE) +
REPLACE +
DESCR('REMOTE DEFN OF SEND QUEUE
FROM CAPC TO APPA') +
PUT(ENABLED) +
XMITQ(QMA.XMITQ)  +
RNAME(CAPC.TO.APPA.RECVQ) +
RQMNAME(QMA) +
DEFPSIST(YES)
```

```
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMA') +
XMITQ(QMA.XMITQ) +
CONNAME('127.0.0.1(1450)')

DEFINE CHANNEL(QMA.TO.QMC) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM
QMA')
```

From CLP-B, run the file as:

```
$ runmqsc QMC < SYSC_QMC_MQDEFS_UNI_CA.TXT
```

Starting the Listeners

Start the Listeners on QMA, QMB, and QMC as described in the *P2P three-way replication – The WebSphere MQ layer – Starting the Listeners* section.

Starting the Channels

We need to start the Channels between QMA/QMB and between QMA/QMC.

Start the Channels between QMA and QMB as described in the *P2P three-way replication—The WebSphere MQ layer—Starting the Channels* section.

Start the Channels between QMA and QMC as described in the *P2P three-way replication—The WebSphere MQ layer—Starting the Channels* section.

Note that we do NOT start the Channels between QMB and QMC.

Testing the WebSphere MQ layer

Now that everything is started, we need to test the MQ layer.

We will start by putting test messages onto each system using the `amqsput` command and then retrieving them using the `amqsget` command.

We need to perform two unidirectional tests—between `QMA` and `QMB`, and between `QMA` and `QMC`.

To test unidirectional messages between `QMA` and `QMB`—follow the instructions in the *Unidirectional replication – The WebSphere MQ layer – Testing the WebSphere MQ layer* section.

The put message batch file for `QMA` to `QMC` is `SYSA_QMA_TESTP_UNI_AC.BAT`:

```
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput" CAPA.TO.APPC.
SENDQ.REMOTE     QMA  < SYSA_QMA_TEST1.TXT
```

From CLP-A, run the file as:

```
$ SYSA_QMA_TESTP_UNI_AC.BAT
```

The put message batch file for `QMC` to `QMA` is `SYSC_QMC_TESTP_UNI_CA.BAT`:

```
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPA.ADMINQ.
REMOTE     QMC < SYSB_QMB_TEST2.TXT
```

Note that we are using `SYSB_QMB_TEST2.TXT` as the input text file, but that is valid—it only contains the word `test2`.

From CLP-C, run the file as:

```
$ SYSC_QMC_TESTP_UNI_CA.BAT
```

The get message file (`SYSC_QMC_TESTG_UNI_AC.BAT`) for `QMC` from `QMA` is:

```
echo The amqsget program take 15 seconds to run
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPA.TO.APPC.
RECVQ QMC
@ECHO You should see above:  test1
```

From CLP-C, run the file as:

```
$ SYSC_QMC_TESTG_UNI_AC.BAT
```

The get message file (`SYSA_QMA_TESTG_UNI_CA.BAT`) for `QMA` from `QMC` is:

```
echo The amqsget program take 15 seconds to run
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPA.ADMINQ QMA
@ECHO You should see above:  test2
```

From CLP-A, run the file as:

```
$ SYSA_QMA_TESTG_UNI_CA.BAT
```

Provided we see the messages that we are told we should see, then we have successfully tested the WebSphere MQ layer.

We have now defined the database and WebSphere MQ layers, and can proceed to the Q replication layer.

The Q replication layer

The following sections give the ASNCLP commands to create the control tables, the Replication Queue Maps, and the Q subscription. The tasks are:

- Creating the Q Capture tables on `DB2A`
- Creating the Q Apply tables on `DB2B`
- Creating the Q Apply tables on `DB2C`
- Creating a Replication Queue Map for `DB2A` to `DB2B`
- Creating a Replication Queue Map for `DB2A` to `DB2C`
- Creating the two Q subscriptions

Creating Q Capture control tables on DB2A

Follow the instructions in the *Unidirectional replication – The Q replication layer – Creating Q Capture control tables on DB2A* section.

Creating Q Apply control tables on DB2B

Follow the instructions in the *Unidirectional replication – The Q replication layer – Creating Q Apply control tables on DB2B* section.

Creating Q Apply control tables on DB2C

Follow the instructions in the *Unidirectional replication – The Q replication layer – Creating Q Apply control tables on DB2B* section.

Creating a Replication Queue Map from DB2A to DB2B

Follow the instructions in the *Bidirectional replication – The Q replication layer – Creating a Replication Queue Map for DB2A to DB2B* section.

Creating a Replication Queue Map from DB2A to DB2C

Follow the instructions in the *P2P three-way replication – The Q replication layer – Creating a Replication Queue Map for DB2A to DB2C* section.

Creating a Q subscription(s)

We need to create two unidirectional Q subscriptions – one from `A` to `B` and one `A` to `C`.

For the Q subscription from `A` to `B`, refer to the *Unidirectional replication – The Q replication layer – Creating a Q subscription* section.

The file to create a unidirectional Q subscription from `A` to `C` is called `SYSA_crt_qsub_uni_AC.asnclp`:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2A;
SET SERVER TARGET  TO DB DB2C;

SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY   SCHEMA        ASN;

SET QMANAGER QMA FOR CAPTURE SCHEMA;
SET QMANAGER QMC FOR APPLY   SCHEMA;

CREATE QSUB
USING REPLQMAP RQMA2C
(SUBNAME TAB1 ERIC.T1
OPTIONS
HAS LOAD PHASE I
SPILL_MODELQ "IBMQREP.SPILL.MODELQ"
TARGET NAME FRED.T1
TYPE USERTABLE
KEYS (C1)
CONFLICT ACTION F
LOAD TYPE 0);
```

From CLP-A, issue:

```
$ asnclp -f SYSA_crt_qsub_uni_AC.asnclp
```

Starting Q Capture and Q Apply

Now we need to start Q Capture and Q Apply.

Starting Q Capture on DB2A

To start Q Capture on `DB2A`, follow the instructions in the *Unidirectional replication – Starting Q Capture and Q Apply – Starting Q Capture on DB2A* section.

Starting Q Apply on DB2B and DB2C

To start Q Apply on `DB2B`, follow the instructions in the *Unidirectional replication – Starting Q Capture and Q Apply – Starting Q Apply on DB2B* section.

From CLP-C, issue:

```
$ start asnqapp APPLY_SERVER=db2c
```

Testing replication

We are now in a position to test the unidirectional replication from one source to two targets. Insert a record into `ERIC.T1` on `DB2A` and check that it is replicated to `FRED.T1` on `DB2B` and `DB2C`.

From CLP-A, issue:

```
$ db2 "insert into eric.t1 values (1,1,'H') "
```

From CLP-B, issue:

```
$ "db2 select * from fred.t1 "

C1          C2          C3
----------- ----------- ----------
          1           1 H
  1 record(s) selected.
```

We should see one record in `FRED.T1` on `DB2B`.

From CLP-C, issue:

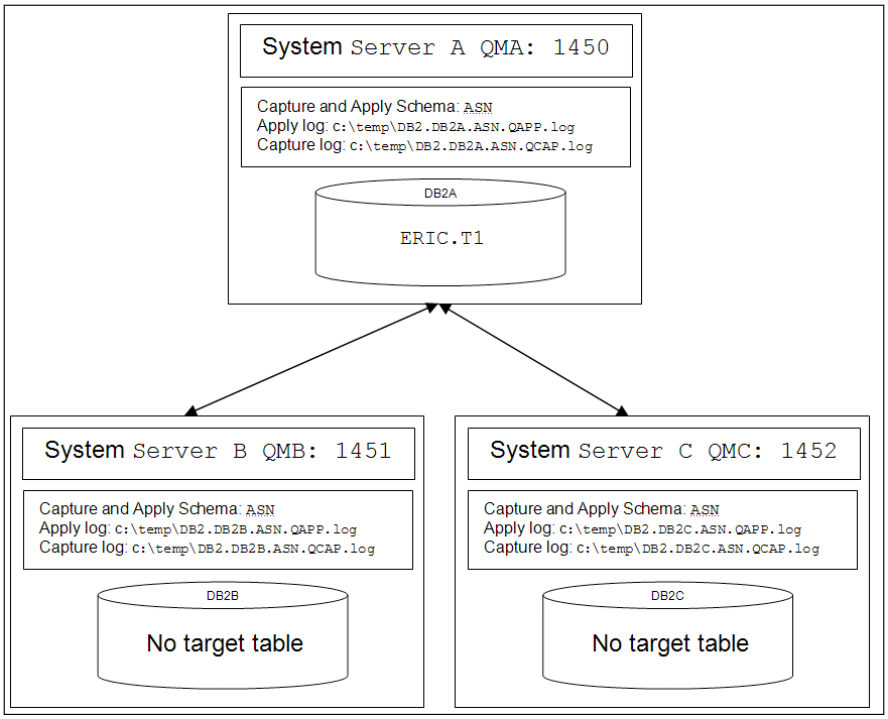
```
$ db2 "select * from fred.t1 "

C1          C2          C3
----------- ----------- ----------
          1           1 H
  1 record(s) selected.
```

We should see one record in `FRED.T1` on `DB2C`.

We can see that the unidirectional Q replication setup from one source to two targets is working.

Bidirectional replication to two targets (B-tree)

The following shows the setup that we will use:



The B-tree structure was discussed in the *The different types of Q replication – Tree replication* section of *Chapter 1*.

The test table `ERIC.T1` only exists on `DB2A`—Q Apply will create the test table `FRED.T1` on the other databases.

The database layer

We need to create three source databases called `DB2A`, `DB2B`, and `DB2C`—follow the instructions in the *First steps – Database creation* section.

We now have the database layer defined and can now proceed to the WebSphere MQ layer.

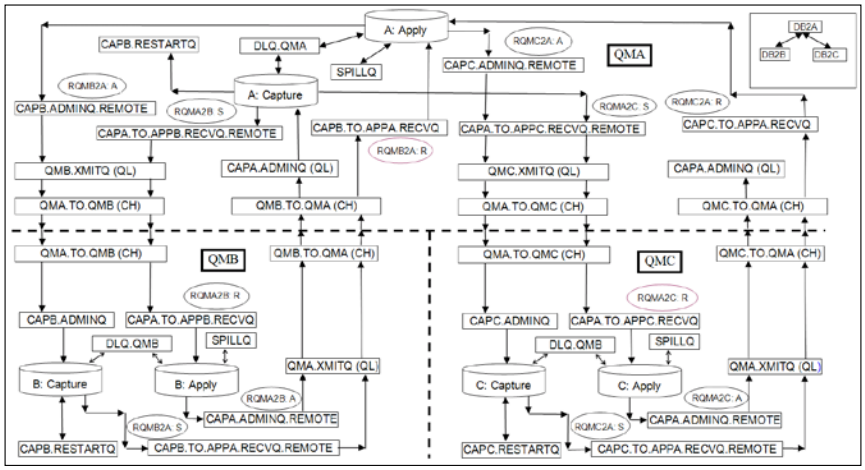
The WebSphere MQ layer

This section deals with the WebSphere MQ layer, which covers creating the Queue Managers and the appropriate queues, and then starting the Listeners and Channels.

Creating the Queue Managers and Queues

We need to create and start three Queue Managers called `QMA`, `QMB`, and `QMC`—follow the instructions in the *First steps – Queue Manager processing* section.

The following diagram shows the queues needed for bidirectional replication to two targets. What we are doing here is effectively setting up bidirectional replication between `DB2A` and `DB2B` on the one hand, and `DB2A` and `DB2C` on the other. How the queues are related is shown in the following diagram:



The values for the Send, Receive, and Administration Queues for each RQM are shown in the following

	RQMA2B	**RQMB2A**
SENDQ	CAPA.TO.APPB.RECVQ.REMOTE on QMA	CAPB.TO.APPA.RECVQ.REMOTE on QMB
RECVQ	CAPA.TO.APPB.RECVQ on QMB	CAPB.TO.APPA.RECVQ on QMA
ADMINQ	CAPA.ADMINQ.REMOTE on QMB	CAPB.ADMINQ.REMOTE on QMA
	RQMA2C	**RQMC2A**
SENDQ	CAPA.TO.APPC.RECVQ.REMOTE on QMA	CAPC.TO.APPA.RECVQ.REMOTE on QMC
RECVQ	CAPA.TO.APPC.RECVQ on QMC	CAPC.TO.APPA.RECVQ on QMA
ADMINQ	CAPA.ADMINQ.REMOTE on QMC	CAPC.ADMINQ.REMOTE on QMA

To set up the bidirectional queues for QMA and QMB, refer to the *Bidirectional replication – The WebSphere MQ layer – Creating the Queue Managers and queues* section.

The queues we need for replication between DB2A and DB2C to be run against QMA are in SYSA_QMA_MQDEFS_BIP2P2W_AC.TXT file:

```
DELETE QLOCAL(CAPA.ADMINQ) PURGE
DEFINE QLOCAL(CAPA.ADMINQ) +
REPLACE +
DESCR('LOCAL DEF OF ADMINQ FOR
CAPA') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DELETE QLOCAL(CAPA.RESTARTQ)
PURGE
DEFINE QLOCAL(CAPA.RESTARTQ) +
REPLACE +
DESCR('LOC DEF OF RESTART FOR
CAPA') +
DEFPSIST(YES)

DEFINE QLOCAL(CAPC.TO.APPA.
RECVQ) +
REPLACE +
DESCR('LOCAL RECV Q -APPA FROM
CAPC') +
PUT(ENABLED) +
GET(ENABLED) +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QREMOTE(CAPC.ADMINQ.
REMOTE) +
REPLACE +
DESCR('RMT DEF OF ADMINQ FOR
CAPC') +
PUT(ENABLED) +
```

```
+
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QMODEL(IBMQREP.SPILL.
MODELQ) +
REPLACE +
DEFSOPT(SHARED) +
MAXDEPTH(500000) +
MSGDLVSQ(FIFO) +
DEFTYPE(PERMDYN)
*
DEFINE QLOCAL(DEAD.LETTER.QUEUE.
QMA) +
REPLACE +
DESCR('LOCAL DEAD LETTER QUEUE
QMA') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)
*
DEFINE QREMOTE(CAPA.TO.APPC.
SENDQ.REMOTE) +
REPLACE +
XMITQ(QMC.XMITQ) +
RNAME(CAPC.ADMINQ) +
RQMNAME(QMC) +
DEFPSIST(YES)
*
DEFINE QLOCAL(QMC.XMITQ) +
REPLACE +
DESCR('TRANSMISSION QUEUE TO
QMC') +
USAGE(XMITQ) +
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
TRIGTYPE(FIRST) +
TRIGDATA(QMA.TO.QMC) +
INITQ(SYSTEM.CHANNEL.INITQ)
*
DEFINE CHANNEL(QMA.TO.QMC) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMC') +
XMITQ(QMC.XMITQ) +
CONNAME('127.0.0.1(1452)')
*
DEFINE CHANNEL(QMC.TO.QMA) +
CHLTYPE(RCVR) +
```

```
DESCR('RMT DEF SND Q CAPA TO APPC') +
PUT(ENABLED) +
XMITQ(QMC.XMITQ)  +
RNAME(CAPA.TO.APPC.RECVQ) +
RQMNAME(QMC) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM QMC')
*
```

From CLP-A, run the file as:

```
$ runmqsc QMA < SYSA_QMA_MQDEFS_BIP2P2W_AC.TXT
```

The queues we need for replication between `DB2C` and `DB2A` to be run against `QMC` are in `SYSC_QMC_MQDEFS_BIP2P2W_CA.TXT` file:

```
DELETE QLOCAL(CAPC.ADMINQ) PURGE
DEFINE QLOCAL(CAPC.ADMINQ) +
REPLACE +
DESCR('LOC DEF OF ADMINQ FOR CAPC') +
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)

DELETE QLOCAL(CAPC.RESTARTQ) PURGE
DEFINE QLOCAL(CAPC.RESTARTQ) +
REPLACE +
DESCR('LOC DEF OF RESTART CAPC') +
PUT(ENABLED) +
GET(ENABLED) +
DEFPSIST(YES)

DEFINE QLOCAL(CAPA.TO.APPC.RECVQ) +
REPLACE +
DESCR('LOCAL RECEIVE QUEUE') +
PUT(ENABLED) +
GET(ENABLED) +
DEFSOPT(SHARED) +
DEFPSIST(YES)

DEFINE QREMOTE(CAPA.ADMINQ.REMOTE) +
REPLACE +
DESCR('RMT DEF OF ADMINQ FOR CAPA') +
PUT(ENABLED) +
XMITQ(QMA.XMITQ) +
```

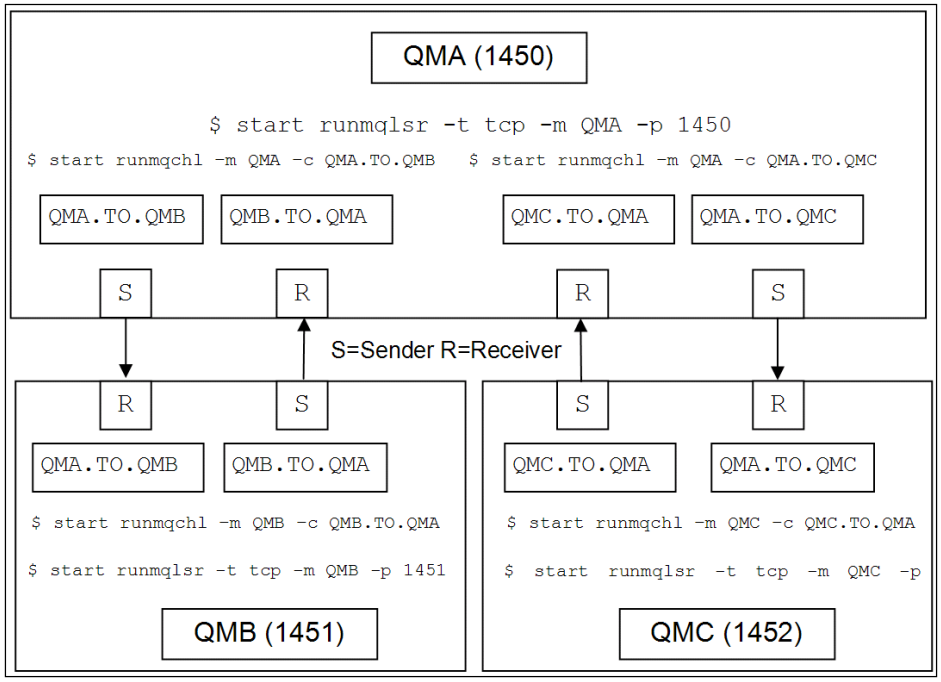
```
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)


DEFINE QMODEL(IBMQREP.SPILL.
MODELQ) +
REPLACE +
DEFSOPT(SHARED) +
MAXDEPTH(500000) +
MSGDLVSQ(FIFO) +
DEFTYPE(PERMDYN)


DEFINE QLOCAL(DEAD.LETTER.QUEUE.
QMC)+
REPLACE +
DESCR('LOCAL DEAD LETTER Q QMC')
+
PUT(ENABLED) +
GET(ENABLED) +
SHARE +
DEFSOPT(SHARED) +
DEFPSIST(YES)


DEFINE QREMOTE(CAPC.TO.APPA.
SENDQ.REMOTE)+
REPLACE +
DESCR('RMT DEF SND Q CAPC TO
APPA') +
PUT(ENABLED) +
XMITQ(QMA.XMITQ)  +
RNAME(CAPC.TO.APPA.RECVQ) +
RQMNAME(QMA) +
RNAME(CAPA.ADMINQ) +
RQMNAME(QMA) +
DEFPSIST(YES)


DEFINE QLOCAL(QMA.XMITQ) +
REPLACE +
DESCR('TRANSMISSION QUEUE TO
QMA') +
USAGE(XMITQ) +
PUT(ENABLED) +
GET(ENABLED) +
TRIGGER +
TRIGTYPE(FIRST) +
TRIGDATA(QMC.TO.QMA) +
INITQ(SYSTEM.CHANNEL.INITQ)


DEFINE CHANNEL(QMC.TO.QMA) +
CHLTYPE(SDR) +
REPLACE +
TRPTYPE(TCP) +
DISCINT(0) +
DESCR('SENDER CHANNEL TO QMA') +
XMITQ(QMA.XMITQ) +
CONNAME('127.0.0.1(1450)')


DEFINE CHANNEL(QMA.TO.QMC) +
CHLTYPE(RCVR) +
REPLACE +
TRPTYPE(TCP) +
DESCR('RECEIVER CHANNEL FROM
QMA')
```

From CLP-C, run the file as:

```
$ runmqsc QMC < SYSC_QMC_MQDEFS_BIP2P2W_CA.TXT
```

Starting the Listeners

The following diagram shows the related Listeners and Channels:



Start the Listeners for QMA, QMB, and QMC as described in the *P2P three-way replication – The WebSphere MQ layer – Starting the Listeners* section.

Starting the Channels

For the Channels between QMA and QMB—follow the instructions in the *Unidirectional replication – Starting the Channels* section.

For the Channels between QMA and QMC—follow the instructions in the *P2P three-way replication – The WebSphere MQ layer – Starting the Channels* section.

Testing the WebSphere MQ layer

Now that everything is started, we need to test the MQ layer.

To test bidirectional message replication between QMA and QMB—follow the instructions in the *Bidirectional replication – The WebSphere MQ layer – Testing the WebSphere MQ layer* section.

To test bidirectional message replication between QMA and QMC—follow the instructions.

The put message batch file for QMA is called SYSA_QMA_TESTP_BIP2P2W_AC.BAT:

```
echo The amqsget program take 15 seconds to run

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPA.TO.APPC.
SENDQ.REMOTE     QMA <QMA_TEST1.TXT

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPC.ADMINQ.
REMOTE     QMA <QMA_TEST2.TXT
```

From CLP-A, run the file as:

```
$ SYSA_QMA_TESTP_BIP2P2W_AC.BAT
```

The put message batch file for QMC is called SYSC_QMC_TESTP_BIP2P2W_CA.BAT:

```
echo The amqsget program take 15 seconds to run

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPC.TO.APPA.
SENDQ.REMOTE     QMC <SYSC_QMC_TEST9.TXT


call "C:\Program Files\IBM\WebSphere MQ\bin\amqsput"  CAPA.ADMINQ.
REMOTE     QMC <SYSC_QMC_TEST10.TXT
```

From CLP-C, run the file as:

```
$ SYSC_QMC_TESTP_BIP2P2W_CA.BAT
```

Once we have put the test messages onto each system, we can retrieve them.

The get message batch file for QMA from QMC is SYSA_QMA_TESTG_BIP2P2W_AC.BAT:

```
@echo The amqsget program take 15 seconds to run

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPA.ADMINQ QMA

@ECHO You should see above:  test10

call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPC.TO.APPA.
RECVQ QMA

@ECHO You should see above:  test9
```

From CLP-A, run the file as:

```
$ SYSA_QMA_TESTG_BIP2P2W_AC.BAT
```

The get message batch file for QMC from QMA is SYSC_QMC_TESTG_BIP2P2W_CA.BAT:

```
@echo The amqsget program take 15 seconds to run
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPA.TO.APPC.
RECVQ QMC
@ECHO You should see above:  test1
call "C:\Program Files\IBM\WebSphere MQ\bin\amqsget" CAPC.ADMINQ QMC
@ECHO You should see above:  test2
```

From CLP-C, run the file as:

```
$ SYSC_QMC_TESTG_BIP2P2W_CA.BAT
```

Provided we see the messages that we are told we should see, then we have successfully tested the WebSphere MQ layer.

We have now defined the database and WebSphere MQ layers, and can proceed to the Q replication layer.

The Q replication layer

The following sections give the ASNCLP commands to create the control tables, the Replication Queue Maps, and the Q subscription. The tasks are:

- Creating the Q Capture and Q Apply control tables on DB2A
- Creating the Q Capture and Q Apply control tables on DB2B
- Creating the Q Capture and Q Apply control tables on DB2C
- Creating a Replication Queue Map for DB2A to DB2B
- Creating a Replication Queue Map for DB2A to DB2C
- Creating a Replication Queue Map for DB2B to DB2A
- Creating a Replication Queue Map for DB2C to DB2A
- Creating a Q subscription

Creating Q Capture/Q Apply control tables on DB2A

Follow the instruction in the *Bidirectional replication – The Q replication layer – Creating Q Capture/Q Apply control tables on DB2A* section.

Creating Q Capture/Q Apply control tables on DB2B

Follow the instruction in the *Bidirectional replication – The Q replication layer – Creating Q Capture/Q Apply control tables on DB2B* section.

Creating Q Capture/Q Apply control tables on DB2C

Follow the instruction in the *P2P three-way replication – The Q replication layer – Creating Q Capture/Q Apply control tables on DB2C* section.

Creating a Replication Queue Map for DB2A to DB2B

Follow the instruction in the *Bidirectional replication – The Q replication layer – Creating a Replication Queue Map for DB2A to DB2B* section.

Creating a Replication Queue Map for DB2A to DB2C

Follow the instruction in the *P2P three-way replication – The Q replication layer – Creating a Replication Queue Map for DB2A to DB2C* section.

Creating a Replication Queue Map for DB2B to DB2A

Follow the instruction in the *P2P three-way replication – The Q replication layer – Creating a Replication Queue Map for DB2B to DB2A* section.

Creating a Replication Queue Map for DB2C to DB2A

Follow the instruction in the *P2P three-way replication – The Q replication layer – Creating a Replication Queue Map for DB2C to DB2A* section.

Creating the Q subscriptions

To create a bidirectional Q subscription between `DB2A` and `DB2B`, follow the instructions in the *Bidirectional replication – The Q replication layer – Creating a bidirectional Q subscription* section .

The two files needed for the bidirectional Q subscription between `DB2A` and `DB2C` are `SYSA_loadbidi_AC.asnclp` and `SYSA_contbidi_AC.txt`. `SYSA_loadbidi_AC.asnclp` file contains:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;
LOAD MULTIDIR REPL SCRIPT "SYSA_contbidi_AC.txt";
```

File `SYSA_contbidi_AC.txt` contains:

```
set subgroup "TABT1AC";

set server multidir to db "DB2A";
set server multidir to db "DB2C";
set multidir schema "DB2A".ASN;
set multidir schema "DB2C".ASN;

set connection SOURCE "DB2A".ASN TARGET "DB2C".ASN replqmap "RQMA2C";
set connection SOURCE "DB2C".ASN TARGET "DB2A".ASN replqmap "RQMC2A";
set tables("DB2A".ASN.ERIC.T1, "DB2C".ASN.FRED.T1);
CREATE QSUB subtype b;
```

From CLP-A:

```
$ asnclp -f SYSA_loadbidi_AC.asnclp
```

Starting Q Capture and Q Apply

Now we need to start Q Capture and Q Apply.

Starting Q Capture on DB2A, DB2B, and DB2C

To start Q Capture, follow the instructions in the *Q Capture administration – Starting Q Capture* section of *Chapter 6, Administration Tasks*.

Wait for all Q Captures to be up and running before starting the Q Applys.

Starting Q Apply on DB2A, DB2B, and DB2C

To start Q Apply, follow the instructions in the *Q Apply administration – Starting Q Apply* section of *Chapter 6*.

Issuing a CAPSTART command

Follow the instructions in the *P2P three-way—Starting Q Capture and Q Apply—Issuing a CAPSTART command* section.

Testing replication

We are now in a position to test our bidirectional replication to two targets setup. We will insert a record on `DB2A` and check that it has replicated to `DB2B` and `DB2C`.

From CLP-A, issue:

```
$ db2 "insert into eric.t1 values (1,1,'H') "
$ db2 "select * from eric.t1 "
C1          C2          C3
----------- ----------- ----------
          1           1 H
```

From CLP-C, issue:

```
$ db2 "select * from fred.t1 "
C1          C2          C3
----------- ----------- ----------
          1           1 H
```

From CLP-B, issue:

```
$ db2 "select * from fred.t1 "
C1          C2          C3
----------- ----------- ----------
          1           1 H

$ db2 "insert into fred.t1 values (2,2,'H') "
```

From CLP-A, issue:

```
$ db2 "select * from eric.t1 "
C1          C2          C3
----------- ----------- ----------
          1           1 H
          2           2 H
```

From CLP-C, issue:

```
$ db2 "select * from fred.t1 "
C1          C2          C3
----------- ----------- ----------
          1           1 H
          2           2 H
$ db2 "insert into fred.t1 values (3,3,'H') "
```

From CLP-A, issue:

```
$ db2 "select * from eric.t1 "
C1           C2           C3
------------ ------------ ----------
           1            1 H
           2            2 H
           3            3 H
```

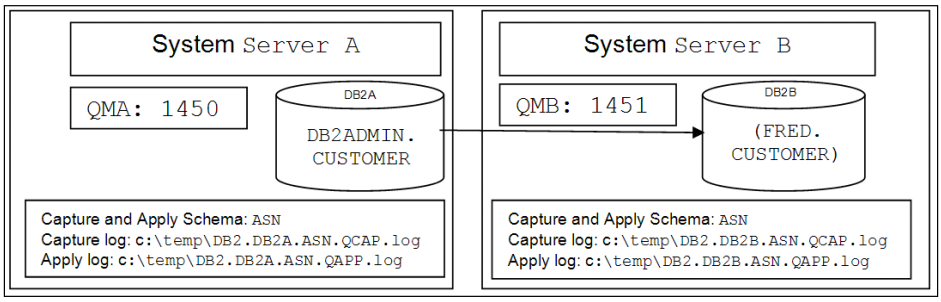
From CLP-B, issue:

```
$ db2 "select * from fred.t1 "
C1           C2           C3
------------ ------------ ----------
           1            1 H
           2            2 H
           3            3 H
```

We can see that the bidirectional B-tree structure Q replication setup from one source to two targets is working.

Unidirectional replication for an XML data type

The following diagram shows the setup we will use:



We will start with a test table `DB2ADMIN.CUSTOMER`, which only exists on `DB2A`—Q Apply will create its equivalent called `FRED.CUSTOMER` on `DB2B`.

There are a few restrictions concerning replicating the XML data type:

- We cannot filter rows based on the contents of an XML document.
- The documents are not validated by Q Apply.
- We cannot replicate the XML schema registrations.
- We cannot replicate XML columns from Oracle sources.
- User-defined unique indexes on XPATH expressions on XML columns are not supported. If a unique index exists, then when we try and create the Q subscription, we will get `ASN0999E Q Replication does not support the XML column DB2ADMIN.CUSTOMER.INFO because it has one or more unique indexes." : "" : Error condition "", error code(s): "", "", ""." message. To replicate these columns, drop the unique index and use a non-unique index.`

The database layer

We need to create a source database called `DB2A` and a target database called `DB2B`—because we are going to use the XML sample database as the source, we will not follow the instructions in the *First steps – Database creation* section, but use the following commands.

From CLP-A, issue:

```
$ db2sampl –name db2a -xml
$ db2 UPDATE DB CFG FOR db2a USING logarchmeth1 disk:c:\temp
$ db2 BACKUP DB db2a TO c:\temp
```

Now we can check if we have any indexes on the XML data type column using the following SQL:

```
$ db2 "select substr(indschema,1,10) as indschema, substr(indname,1,20)
as indname,substr(colnames,1,20) as colnames from syscat.indexes where
tabname = 'CUSTOMER'"
```

And we can drop the indexes as follows:

```
$ db2 DROP INDEX db2admin.cust_cid_xmlidx
$ db2 drop index DB2ADMIN.cust_name_xmlidx
```

From CLP-B, issue:

```
$ db2 CREATE DB db2b
```

We have now completed the database layer and can proceed to the WebSphere MQ layer.

The WebSphere MQ layer

This section deals with the WebSphere MQ layer, which covers creating the Queue Managers and the appropriate queues, and then starting the Listeners and Channels.

Creating the Queue Managers and Queues

We need to create two Queue Managers called `QMA` and `QMB`—follow the instructions in the *Create/start/stop a Queue Manager* section of *Chapter 4*, *WebSphere MQ for the DBA*.

Create the queues as described in the *Unidirectional replication – The WebSphere MQ layer – Creating the Queue Managers and queues* section.

Starting the Listeners

Start the Listeners for `QMA` and `QMB` as described in the *Bidirectional replication – The WebSphere MQ layer – Starting the Listeners* section.

Starting the Channels

Start the Channels between `QMA` and `QMB` as described in the *Bidirectional replication – The WebSphere MQ layer – Starting the Channels* section.

Testing the WebSphere MQ layer

Now that everything is started, we need to test the MQ layer.

Test the WebSphere layer as described in the *Unidirectional replication – The WebSphere MQ layer – Testing the WebSphere MQ layer* section.

Provided we see the messages that we are told we should see, then we have successfully tested the WebSphere MQ layer.

We have now defined the database and WebSphere MQ layers, and can proceed to the Q replication layer.

The Q replication layer

The following sections give the ASNCLP commands to create the control tables, the Replication Queue Maps, and the Q subscription. The tasks are:

- Creating the Q Capture tables on `DB2A`
- Creating the Q Apply tables on `DB2B`
- Creating a Replication Queue Map for `DB2A` to `DB2B`
- Creating a Q subscription

Creating Q Capture control tables on DB2A

Create the Q Capture control tables as described in the *Unidirectional replication – The Q replication layer – Creating Q Capture control tables on DB2A* section.

Creating Q Apply control tables on DB2B

Create the Q Apply control tables as described in the *Unidirectional replication – The Q replication layer – Create Q Apply control tables on DB2B* section.

Creating a Replication Queue Map from DB2A to DB2B

Create a Replication Queue Map as described in the *Unidirectional replication – The Q replication layer – Creating a Replication Queue Map from DB2A to DB2B* section.

Creating a Q subscription

The ASNCLP command file to create a unidirectional Q subscription is called `SYSA_crt_qsub_uni_AB.asnclp` and is described in the *Q subscription maintenance – Q subscription for unidirectional replication* section of *Chapter 5*, *The ASNCLP Command Interface*.

For this test change the source and target tables to be `CUSTOMER`. Therefore edit the `SYSA_crt_qsub_uni_AB.asnclp` file, make the change to the source and target table names, and save the file as `SYSA_crt_qsub_uni_AB_XML.asnclp`:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2A;
SET SERVER TARGET  TO DB DB2B;

SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY   SCHEMA        ASN;

SET QMANAGER QMA FOR CAPTURE SCHEMA;
SET QMANAGER QMB FOR APPLY   SCHEMA;

CREATE QSUB
USING REPLQMAP RQMA2B
(SUBNAME TAB1 DB2ADMIN.CUSTOMER
OPTIONS
HAS LOAD PHASE I
SPILL_MODELQ "IBMQREP.SPILL.MODELQ"
TARGET NAME FRED.CUSTOMER
TYPE USERTABLE
```

```
KEYS (CID)
CONFLICT ACTION F
LOAD TYPE 2);
```

From CLP-A, issue:

```
$ asnclp -f SYSA_crt_qsub_uni_AB_XML.asnclp
```

Starting Q Capture and Q Apply

Now we need to start Q Capture and Q Apply.

Starting Q Capture on DB2A

Start Q Capture as described in the *Unidirectional replication – Starting Q Capture and Q Apply – Starting Q Capture on DB2A* section.

Starting Q Apply on DB2B

Start Q Apply as described in the *Unidirectional replication – Starting Q Capture and Q Apply – Starting Q Apply on DB2B* section.

Testing replication

There are many tests that we could perform, following are just a couple.

The first test we will perform is to add an element to a particular record in the `DB2ADMIN.CUSTOMER` table on `DB2A` and check that it is replicated to `FRED.CUSTOMER` on `DB2B`. We will use the following SQL (`trans04_xml.sql`):

From CLP-A, issue:

```
$ db2 "select info from db2admin.customer where cid=1003 "
```

```
<customerinfo xmlns="http://posample.org" Cid="1003"><name>Robert
Shoemaker</name><addr country="Canada"><street>1596 Baseline</
street><city>Aurora</city><prov-state>Ontario</prov-state><pcode-
zip>N8X 7F8</pcode-zip></addr><phone type="work">905-555-7258</
phone><phone type="home">416-555-2937</phone><phone
type="cell">905-555-8743</phone><phone type="cottage">613-555-3278</
phone></customerinfo>
```

The SQL statement to add an element is an `UPDATE` statement as shown next:

```
UPDATE
db2admin.customer
SET info = (select xmlquery('declare default element namespace
"http://posample.org";
transform
```

```
   copy $newinfo := $info
   modify do insert <status>Current</status> as last into $newinfo/
   customerinfo
   return $newinfo' passing info as "info")
   FROM customer WHERE customer.cid = 1003 )
   WHERE cid = 1003 #
```

```
$ db2 -td# -vf trans04_xml.sql
```

```
$ db2 "select info from db2admin.customer where cid=1003"
```

```
<customerinfo xmlns="http://posample.org" Cid="1003"><name>Robert
Shoemaker</name><addr country="Canada"><street>1596 Baseline</
street><city>Aurora</city><prov-state>Ontario</prov-state><pcode-
zip>N8X 7F8</pcode-zip></addr><phone type="work">905-555-7258</
phone><phone type="home">416-555-2937</phone><phone
type="cell">905-555-8743</phone><phone type="cottage">613-555-3278</
phone><status>Current</status></customerinfo>
```

We can see that the XML record has been altered on `DB2A`.

Now let's check that the change has been replicated to `DB2B`.

From CLP-B, issue:

```
$ db2 "select info from fred.customer where cid=1003"
```

```
<customerinfo xmlns="http://posample.org" Cid="1003"><name>Robert
Shoemaker</name><addr country="Canada"><street>1596 Baseline</
street><city>Aurora</city><prov-state>Ontario</prov-state><pcode-
zip>N8X 7F8</pcode-zip></addr><phone type="work">905-555-7258</
phone><phone type="home">416-555-2937</phone><phone
type="cell">905-555-8743</phone><phone type="cottage">613-555-3278</
phone><status>Current</status></customerinfo>
```

We can see that the alteration to the XML record has been replicated to `FRED.CUSTOMER` on `DB2B`.

For a second test, we will update an element in an XML column for a particular `Cid` value.

Let's remind ourselves of the structure of the `INFO` column:

```
<customerinfo xmlns="http://posample.org" Cid="1003">
  <name>Robert Shoemaker</name>
  <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Aurora</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N8X 7F8</pcode-zip>
  </addr>
```

```
<phone type="work">905-555-7258</phone>
<phone type="home">416-555-2937</phone>
<phone type="cell">905-555-8743</phone>
<phone type="cottage">613-555-3278</phone>
</customerinfo>
```

Say we want to update the phone number of `Cid 1003`, but which one? Let's update the phone number for work only. The `UPDATE` statement will look like this (`update_xml01.sql`):

```
update db2admin.customer
set info = xmlquery( 'declare default element namespace "http://
posample.org";
transform
copy $new := $INFO
modify do replace value of $new/customerinfo/phone[@type = "work"]
with "123-456-7890"
return $new')
where cid = 1003#
```

And we run the SQL as follows.

From CLP-A, issue:

```
$ db2 -td# -vf update_xml01.sql
```

Let's check that the update has worked:

```
$ db2 "select info from db2admin.customer where cid=1003"
```

```
1003 <customerinfo xmlns="http://posample.org" Cid="1003"><name>Robert
Shoemaker</name><addr country="Canada"><street>1596 Baseline</
street><city>Aurora</city><prov-state>Ontario</prov-state><pcode-
zip>N8X 7F8</pcode-zip></addr><phone type="work">123-456-7890</
phone><phone type="home">416-555-2937</phone><phone
type="cell">905-555-8743</phone><phone type="cottage">613-555-3278</
phone><status>Current</status></customerinfo>
```

We can see that the update has worked on `DB2A`. Now let's check that the replication has worked on `DB2B`.

From CLP-B, issue:

```
$ db2 "select info from fred.customer where cid = 1003"
```

```
1003 <customerinfo xmlns="http://posample.org" Cid="1003"><name>Robert
Shoemaker</name><addr country="Canada"><street>1596 Baseline</
street><city>Aurora</city><prov-state>Ontario</prov-state><pcode-
zip>N8X 7F8</pcode-zip></addr><phone type="work">123-456-7890</
phone><phone type="home">416-555-2937</phone><phone
type="cell">905-555-8743</phone><phone type="cottage">613-555-3278</
phone><status>Current</status></customerinfo>
```

We can see that the `UPDATE` to the XML record has been replicated to `FRED.CUSTOMER` on `DB2B`.

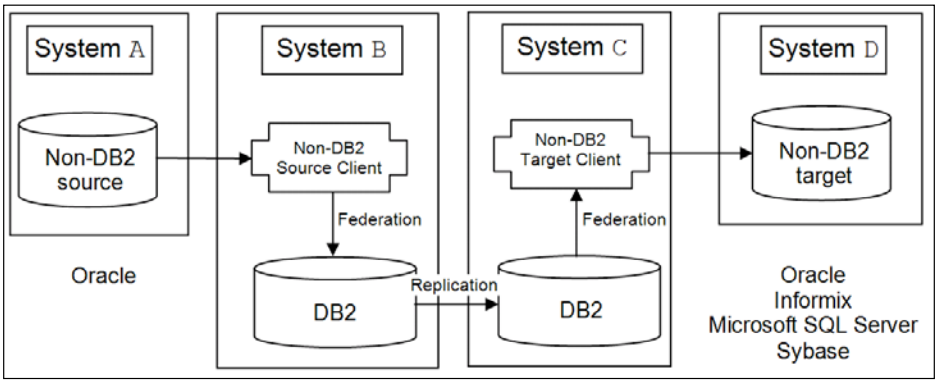
We can see that the replication of XML data in a unidirectional Q replication setup is working.

Federated replication

In this section, we look at federated replication, in particular replicating from DB2 to Oracle and since DB2 9.7.1, replicating from Oracle to DB2. This is particularly useful if you are using the DB2 9.7 Oracle compatibility feature, and want to keep your Oracle and DB2 databases in step. The Oracle log mining feature was added specifically for customers with this need.

Available sources/targets

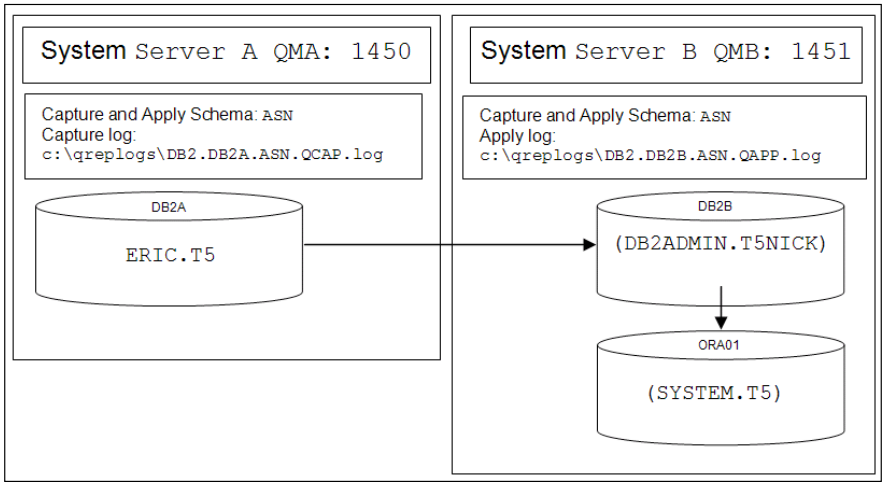
To replicate to a federated target (non-DB2), we need to install the InfoSphere Federation Server code and the target client database code on the server which will run Q Apply, in addition to the replication code. In the following figure, we are Replicating from a non-DB2 relational source (Oracle) to a non-DB2 relational target.



A table listing the restrictions that SQL Replication and Q replication have different restrictions on the source and target can be found in the DB2 Information Center, by searching for *Comparison of sources and targets in SQL replication and Q replication*: `http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.swg.im.iis.db.repl.intro.doc/topics/iiyrcintdifst.html`

Replicating from DB2 to Oracle

The following diagram shows the setup we will use:



The Listening port for `QMA` is `1450` and for `QMB` is `1451`.

The process of replicating to a non-DB2 database involves using a federated nickname as a target inside DB2. The components that need to be set up to make this work are as follows:

- **Wrapper**: A translation library specific to th**Server**: The specific database we will be Replicating to.
- **User Mapping**: To match users and passwords on source and target.
- **Nickname**: This points to the target table and will be created automatically by Q Apply.

The test tables `ERIC.T5` only exist on `DB2A`—Q Apply will create the nickname on `DB2B` and the target table on the Oracle system.

We are assuming that Oracle is installed.

The database layer

We need to create two databases, a source database `DB2A` and a target database `DB2B`—follow the instructions in the *First steps – Database creation* section.

Once we have created the source database (DB2A), we need to create the source table called ERIC.T5 as follows:

From CLP-A, issue:

```
$ db2 "create table ERIC.T5(C1 INT NOT NULL PRIMARY KEY, C2 int, C3
char(10)) "
```

What we need to do is set up the federation from DB2B to the Oracle database.

We will only define a wrapper and server for the Oracle target, we will NOT define any nicknames manually—the nickname definition is done at the Q subscription definition stage.

Our first task is to create the Oracle wrapper. The CREATE SERVER statement defines a data source to our federated database (DB2B).

From CLP-B, issue:

```
$ db2 "CREATE WRAPPER NET8 LIBRARY 'db2net8.dll'  "
```

Where NET8 is the Oracle client run time and db2net8.dll is the Windows dll installed in \sqllib\bin to support the distributed transaction. On UNIX systems, the dll is called libdb2net8.dll.

Next, we create a server definition:

```
$ db2 "CREATE SERVER ORA01 TYPE ORACLE VERSION '10g' WRAPPER NET8
OPTIONS( ADD NODE 'ORA01', COLLATING_SEQUENCE 'N') "
```

Next, we create a user mapping definition:

```
$ db2 "CREATE USER MAPPING FOR DB2ADMIN SERVER ORA01 OPTIONS ( ADD
REMOTE_AUTHID 'system', ADD  REMOTE_PASSWORD 'pworc') "
```

We have now completed the database layer and can now proceed to the WebSphere MQ layer.

The WebSphere MQ layer

This section deals with the WebSphere MQ layer, which covers creating the Queue Managers and the appropriate queues, and then starting the Listeners and Channels.

Creating the Queue Managers and Queues

We need to create and start two Queue Managers called QMA and QMB—follow the instructions in the *First steps – Queue Manager processing* section.

Create the queues as described in the *Unidirectional replication – The WebSphere MQ layer – Creating the Queue Managers and queues* section.

Starting the Listeners

Start the Listeners for `QMA` and `QMB` as described in the *Bidirectional replication – The WebSphere MQ layer – Starting the Listeners* section.

Starting the Channels

Start the Channels between `QMA` and `QMB` as described in the *Bidirectional replication – The WebSphere MQ layer – Starting the Channels* section.

Testing the WebSphere MQ layer

Now that everything is started, we need to test the MQ layer.

Test the WebSphere layer as described in the *Unidirectional replication – The WebSphere MQ layer – Testing the WebSphere MQ layer* section.

We have now defined the database and WebSphere MQ layers, and can proceed to the Q replication layer.

The Q replication layer

The following sections give the ASNCLP commands to create the control tables, the Replication Queue Maps, and the Q subscription. The tasks are:

- Creating the Q Capture tables on `DB2A`
- Creating the Q Apply tables on `DB2B`
- Creating a Replication Queue Map for `DB2A` to `DB2B`
- Creating a Q subscription
- Populating the source table

Creating Q Capture control tables on DB2A

Create the Q Capture control tables as described in the *Unidirectional replication – The Q replication layer – Creating Q Capture control tables on DB2A* section.

Creating Q Apply control tables on DB2B

The ASNCLP commands to create the Q Apply control tables are in a file called `SYSB_db2b_crt_apply_oracle.asnclp`:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER TARGET TO DB DB2B NONIBM SERVER ORA01 id "db2admin" password
"pworc" ;

SET QMANAGER QMB FOR APPLY SCHEMA;
SET APPLY SCHEMA ASN;

CREATE CONTROL TABLES FOR APPLY SERVER IN FEDERATED RMT SCHEMA system;
```

From CLP-B, run the file as:

```
$ asnclp -f SYSB_db2b_crt_apply_oracle.asnclp
```

The Q Apply tables control tables are split between `DB2B` and the Oracle database:

On `DB2B`**:**	**On Oracle:**
`ASN.IBMQREP_APPLYENQ`	`ASN.IBMQREP_SAVERI`
`ASN.IBMQREP_APPLYMON`	`ASN.IBMQREP_SPILLEDROW`
`ASN.IBMQREP_APPLYPARMS`	`ASN.IBMQREP_RECVQUEUES`
`ASN.IBMQREP_APPLYTRACE`	`ASN.IBMQREP_TARGETS`
	`ASN.IBMQREP_TRG_COLS`
	`ASN.IBMQREP_SPILLQS`
	`ASN.IBMQREP_EXCEPTIONS`
	`ASN.IBMQREP_DONEMSG`

Creating a Replication Queue Map from DB2A to target

The Replication Queue Map is created using `SYSA_crt_rqma2b2.asnclp` file:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2A id "db2admin" password "pworc";
SET CAPTURE SCHEMA SOURCE ASN;
SET QMANAGER QMA FOR CAPTURE SCHEMA;

SET SERVER TARGET TO DB DB2B NONIBM SERVER ORA01 id "db2admin" password
"pworc";
SET QMANAGER QMB FOR APPLY SCHEMA;
SET APPLY SCHEMA ASN;
```

```
CREATE REPLQMAP RQMA2B2
USING
ADMINQ "CAPA.ADMINQ.REMOTE"
RECVQ  "CAPA.TO.APPB.RECVQ"
SENDQ  "CAPA.TO.APPB.SENDQ.REMOTE"
NUM APPLY AGENTS 3
MEMORY LIMIT 9
ERROR ACTION S
HEARTBEAT INTERVAL 5
MAX MESSAGE SIZE 4;
```

From CLP-A, run the file as:

```
$ asnclp -f SYSA_crt_rqma2b2.asnclp
```

Creating the Q subscription

The Q subscription is created using `SYSA_crt_qsub2.asnclp` file:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO DB DB2A id "DB2ADMIN" password "pworc";
SET SERVER TARGET TO DB DB2B NONIBM SERVER ORA01 id "DB2ADMIN" password
"pworc";
SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY SCHEMA ASN;
SET QMANAGER QMA FOR CAPTURE SCHEMA;
SET QMANAGER QMB FOR APPLY   SCHEMA;

CREATE QSUB USING REPLQMAP RQMA2B2
(SUBNAME FEDSUB ERIC.T5
OPTIONS HAS LOAD PHASE N
TARGET NAME T5 FEDERATED T5NICK );
```

From CLP-A, run the file as:

```
$ asnclp -f SYSA_crt_qsub2.asnclp
```

Note the following:

1. The userIDs are in capitals and in quotes.
2. The source table on `DB2A` is the fully qualified name `ERIC.T5`.
3. The target table on Oracle will be called `SYSTEM.T5` (because as we did not specify a table schema in the `TARGET NAME` line, the schema will default to what `db2admin` was mapped to on `DB2B` during the User Mapping task).

4. The nickname created on DB2B will be called DB2ADMIN.T5NICK. Because we only specified a federated target table name (T5NICK) and did not specify a schema, the schema will default to what we specified in the SET SERVER TARGET line.

Let's check what the preceding ASNCLP actually does. It generates two script/control files called qrepcap.sql and qrepapp.sql, which run against DB2A and DB2B respectively.

The file which runs against DB2A is called qrepcap.sql and contains:

```
-- DatabaseDB2LUOW (DB2A)

-- CONNECT TO DB2A USER XXXX using XXXX;

INSERT INTO ASN.IBMQREP_SUBS (subname, source_owner, source_name,
sendq, subtype, all_changed_rows, before_values, changed_cols_
only, has_loadphase, state, source_node, target_node, options_flag,
suppress_deletes, target_server, target_alias, target_owner, target_
name, target_type, apply_schema) VALUES ('FEDSUB', 'ERIC', 'T5',
'CAPA.TO.APPB.SENDQ.REMOTE', 'U', 'N', 'Y', 'N', 'N', 'N', 0, 0,
'NNNN', 'N', 'DB2B', 'DB2B', 'DB2ADMIN', 'T5NICK', 3, 'ASN');

INSERT INTO ASN.IBMQREP_SRC_COLS (subname, src_colname, is_key, col_
options_flag) VALUES ('FEDSUB', 'C1', 1, 'YNNNNNNNNN');

INSERT INTO ASN.IBMQREP_SRC_COLS (subname, src_colname, is_key, col_
options_flag) VALUES ('FEDSUB', 'C2', 0, 'YNNNNNNNNN');

INSERT INTO ASN.IBMQREP_SRC_COLS (subname, src_colname, is_key, col_
options_flag) VALUES ('FEDSUB', 'C3', 0, 'YNNNNNNNNN');

-- COMMIT;
```

Note the following:

1. We insert into the DB2A tables:

   ```
   ASN.IBMQREP_SUBS
   IBMQREP_SRC_COLS
   ```

   The file which runs against DB2B is called qrepapp.sql and contains:

   ```
   -- DatabaseDB2LUOW (DB2B)
   -- CONNECT TO DB2B USER XXXX using XXXX;
   SET PASSTHRU ORA01;
   CREATE TABLE "SYSTEM"."T5"( C1 NUMBER(10) NOT NULL , C2
   NUMBER(10), C3 CHARACTER(10), PRIMARY KEY(C1));
   COMMIT;
   SET PASSTHRU RESET;
   COMMIT;
   CREATE NICKNAME DB2ADMIN.T5NICK FOR ORA01."SYSTEM"."T5";
   ```

```
ALTER NICKNAME DB2ADMIN.T5NICK ALTER COLUMN C1 LOCAL TYPE INTEGER
ALTER COLUMN C2 LOCAL TYPE INTEGER;
COMMIT;
INSERT INTO ASN.IBMQREP_TARGETS (subname, recvq, source_owner,
source_name, target_owner, target_name, modelq, source_server,
source_alias, target_type, federated_tgt_srvr, state, subtype,
conflict_rule, conflict_action, error_action, source_node, target_
node, load_type, has_loadphase) VALUES ('FEDSUB', 'CAPA.TO.APPB.
RECVQ', 'ERIC', 'T5', 'DB2ADMIN', 'T5NICK', 'IBMQREP.SPILL.
MODELQ', 'DB2A', 'DB2A', 3, 'ORA01', 'I', 'U', 'K', 'I', 'Q', 0,
0, 0, 'N');
INSERT INTO ASN.IBMQREP_TRG_COLS (subname, recvq, target_colname,
source_colname, is_key, target_colNo, mapping_type, src_col_map,
bef_targ_colname) VALUES ('FEDSUB', 'CAPA.TO.APPB.RECVQ', 'C1',
'C1', 'Y', 0, 'R', null, null);
INSERT INTO ASN.IBMQREP_TRG_COLS (subname, recvq, target_colname,
source_colname, is_key, target_colNo, mapping_type, src_col_map,
bef_targ_colname) VALUES ('FEDSUB', 'CAPA.TO.APPB.RECVQ', 'C2',
'C2', 'N', 1, 'R', null, null);
INSERT INTO ASN.IBMQREP_TRG_COLS (subname, recvq, target_colname,
source_colname, is_key, target_colNo, mapping_type, src_col_map,
bef_targ_colname) VALUES ('FEDSUB', 'CAPA.TO.APPB.RECVQ', 'C3',
'C3', 'N', 2, 'R', null, null);
-- COMMIT;
```

2. We use the `SET PASSTHRU` command from `DB2B` to pass thru to the Oracle system to create the target table called `SYSTEM.T5`.
3. We create a nickname called `DB2ADMIN.T5NICK` for the target table `ORA01."SYSTEM"."T5"`.
4. We insert into the following replication control tables on Oracle:

   `ASN.IBMQREP_TARGETS`

   `ASN.IBMQREP_TRG_COLS`

Populating the source table

We are going to insert two records into our source table `ERIC.T5`.
From CLP-A, issue:

```
$ db2 "insert into eric.t5 values (5,1,'J') "
$ db2 "insert into eric.t5 values (6,2,'H') "
```

Starting Q Capture on DB2A

Start Q Capture as described in the *Unidirectional replication – Starting Q Capture and Q Apply – Starting Q Capture on DB2A* section.

Starting Q Apply on DB2B

Start Q Apply as described in the *Unidirectional replication – Starting Q Capture and Q Apply – Starting Q Apply on DB2B* section.

Testing replication

We are now in a position to test our DB2 to Oracle replication. Insert a record into the source table and check that the record is replicated to the Oracle table.

From CLP-A, issue:

```
$ db2 "insert into eric.t5 values (7,1,'H') "
```

From CLP-B, issue:

```
$ db2 "select * from db2admin.T5nick "
C1          C2          C3
----------- ----------- ----------
          7           1 H
  1 record(s) selected.
```

We can see that the record we inserted into DB2 on `DB2A` has now been replicated to the nickname on `DB2B`. Note that we defined the Q subscription as NOT having an initial load—hence the target table is empty.

And just to convince you, let's log onto the Oracle system and check:

From CLP-B, issue:

```
$ sqlplus system/pworc@ora01
SQL*Plus: Release 10.1.0.2.0 - Production on Tue Feb 6 15:30:01 2007
Copyright (c) 1982, 2004, Oracle.  All rights reserved.
Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 -
Production
With the Partitioning, OLAP and Data Mining options
SQL> select * from SYSTEM.T5;

        C1         C2 C3
---------- ---------- ----------
         7          1 H
```

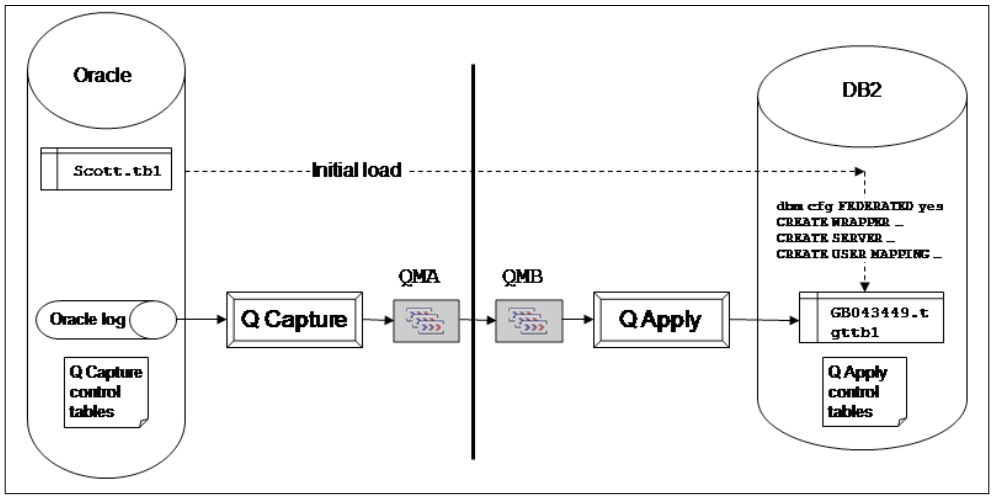
```
SQL> quit

Disconnected from Oracle Database 10g Enterprise Edition Release
10.1.0.2.0 - Production

With the Partitioning, OLAP and Data Mining options

C:\Program Files\IBM\SQLLIB\BIN>
```

We can see that the record we inserted into DB2 on `DB2A` has now been replicated to the Oracle table.

Now let's move on to looking at replicating from Oracle to DB2.

Replicating from Oracle to DB2

In this section, we look at replicating from Oracle to DB2. In previous releases, the method used to replicate from Oracle to DB2 was to create a trigger on the Oracle table and insert any inserted/updated/changed data into a staging table for post processing. In DB2 9.7.1, this method has changed to one where no triggers are required, but Q Capture uses the Oracle log mining capability to detect and extract changes to tables of interest. These changes are then put onto a WebSphere MQ queue for onward transmission to Q Apply. This process is shown in the following diagram:



Before we do any Q replication related work, we need to ensure that the Oracle system is up and running correctly. As part of the work we will do, we need to be able to connect to an Oracle database. We will use the system userid and connect to the `orcl` database using the command:

```
>sqlplus system/passw0rd@orcl
```

The relevant Oracle files in the `C:\oracle\product\10.2.0\db_1\NETWORK\ADMIN` directory are:

- `listener.ora`
- `sqlnet.ora`
- tnsnames.ora

Following is the `listener.ora` file that Oracle generates and following it is the one that we edited for the `orcl` database.

```
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = c:\oracle\product\10.2.0\db_1)
      (PROGRAM = extproc)
    )
  )
LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1))
      (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
    )
  )
```

Edited `listener.ora`:

```
SID_LIST_LISTENER=
  (SID_LIST=
    (SID_DESC=
      (SID_NAME=ORCL)
      (ORACLE_HOME=C:\oracle\product\10.2.0\db_1)
    )
   )

LISTENER=
  (DESCRIPTION_LIST=
    (DESCRIPTION=
      (ADDRESS=(PROTOCOL=IPC)(KEY=EXTPROC1))
      (ADDRESS=(PROTOCOL=TCP)(HOST=IBM-09A0C0E7BD8)(PORT=1521))
    )
  )
```

Following is the `sqlnet.ora` that Oracle generates and following it is the one that we edited.

```
SQLNET.AUTHENTICATION_SERVICES= (NTS)
NAMES.DIRECTORY_PATH= (TNSNAMES, EZCONNECT)
```

Edited `sqlnet.ora`:

```
SQLNET.AUTHENTICATION_SERVICES= (NTS)
NAMES.DIRECTORY_PATH= (TNSNAMES, EZCONNECT)
set oracle_sid=orcl
```

Following is the `tnsnames.ora` that Oracle generates and following it is the one that we edited.

```
ORCL =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = orcl)
    )
  )
EXTPROC_CONNECTION_DATA =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1))
    )
    (CONNECT_DATA =
      (SID = PLSExtProc)
      (PRESENTATION = RO)
    )
  )
```

Edited `tnsnames.ora`:

```
orcl =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = IBM-09A0C0E7BD8)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = orcl)
    )
  )
EXTPROC_CONNECTION_DATA =
  (DESCRIPTION =
```

```
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1))
    )
    (CONNECT_DATA =
      (SID = PLSExtProc)
      (PRESENTATION = RO)
    )
  )
```

We need to set the `ORACLE_HOME` environment variable as:

```
>echo %ORACLE_HOME%

c:\oracle\product\10.2.0\db_1
```

And we also need to add the Oracle jar library into the `CLASSPATH` variable:

```
>echo %CLASSPATH%

C:\Program Files\IBM\SQLLIB\java\db2java.zip;C:\Program Files\IBM\
SQLLIB\java\db2jcc.jar;C:\Program Files\IBM\SQLLIB\java\sqlj.zip;C:\
Program Files\IBM\SQLLIB\java\db2jcc_license_cu.jar;C:\Program Files\
IBM\SQLLIB\bin;C:\Program Files\IBM\SQLLIB\java\common.jar;.;c:\
oracle\product\10.2.0\db_1\oui\jlib\classes12.jar;C:\Program Files\
IBM\WebSphere MQ\Java\lib\com.ibm.mqjms.jar;C:\Program Files\IBM\
WebSphere MQ\Java\lib\com.ibm.mq.jar;.
```

For us to be able to use the Oracle log as a replication source, we need to make the following changes:

- Enable the database to use archive logging
- Enable supplemental logging
- Configure the Oracle LogMiner utility
- Test the java environment for ASNCLP
- Create an ASNCLP Oracle source configuration file
- Switch on supplemental logging for each table that we want to replicate
- Create the Q Apply control tables on `DB2B`
- Create a Replication Queue Map
- Create a Q subscription for the Oracle source

1. Enable the database to use archive logging.

   We can check if the database is already enabled to use archive logging by checking the database view `V$DATABASE` as follows:

   ```
   C:\temp>sqlplus '/ as sysdba'
   ```

Let's check the userid we have connected with:

```
SQL> show user ;
USER is "SYSTEM"
```

To list the Oracle database we are connected to:

```
SQL> SELECT * FROM global_name;
GLOBAL_NAME
----------------------------------
ORCL
```

Now let's check the V$DATABASE database view:

```
SQL> select log_mode from v$database ;
LOG_MODE
------------
NOARCHIVELOG
```

If the log mode shows NOARCHIVELOG, then we need to shutdown the database and alter the database to use archive logging.

If we cannot access the database views, as the following shows:

```
SQL> select log_mode from sys.v$database ;
select * from sys.v$database
                  *
ERROR at line 1:
ORA-00942: table or view does not exist
```

It could mean that ORACLE_SID is not set. We can shutdown the database and alter the database to use archive logging as follows:

```
SQL> shutdown ;
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> STARTUP MOUNT EXCLUSIVE;
ORACLE instance started.
Total System Global Area  535662592 bytes
Fixed Size                  1348508 bytes
Variable Size             260050020 bytes
Database Buffers          268435456 bytes
Redo Buffers                5828608 bytes
```

```
Database mounted.

SQL> ALTER DATABASE ARCHIVELOG;

Database altered.

SQL> ALTER DATABASE OPEN;

Database altered.

SQL> select log_mode from sys.v$database ;

LOG_MODE
------------
ARCHIVELOG
```

We can see from the preceding output that the database is now enabled for archive logging.

2. As we will use the Oracle LogMiner utility, we need to switch on supplemental logging. We can check if supplemental logging is enabled using the following query:

```
SQL> select supplemental_log_data_min from v$database ;

SUPPLEME
--------
NO
```

We can switch on supplemental logging using the following command:

```
SQL> alter database add supplemental log data ;

Database altered.
```

If we check the supplemental logging option again:

```
SQL> select supplemental_log_data_min from v$database ;

SUPPLEME
--------
YES
```

We can see that supplemental logging is enabled.

3. Configure the Oracle LogMiner utility.

We need to create a table space for the LogMiner utility. We should create a file called `c:\temp\create_ora_ts.txt` containing the following:

```
CREATE TABLESPACE logmnrts$
DATAFILE
'c:\oracle\logmnr01.dbf'
SIZE 4M AUTOEXTEND ON
```

```
NOLOGGING EXTENT MANAGEMENT LOCAL;
```

Change directory to `c:\temp`, enter `sqlplus` and run the file as follows:

```
SQL> @create_ora_ts.txt
Tablespace created.
```

Next we need to assign this new table space to the LogMiner utility as follows:

```
SQL> EXECUTE SYS.DBMS_LOGMNR_D.SET_TABLESPACE('logmnrts$');
PL/SQL procedure successfully completed.
```

The final step is to create a LogMiner user account. We need to create a file (`create_asn_account.txt`) containing:

```
create user asn identified by asn
default tablespace users
temporary tablespace temp
profile default account unlock;
grant connect, resource to asn;
grant create session to asn;
alter user asn quota unlimited on logmnrts$;
grant select any transaction to asn;
grant execute_catalog_role to asn;
grant select any table to asn;
grant select on sys.v_$database to asn;
grant select on sys.v_$logmnr_contents to asn;
grant select on sys.v_$logmnr_dictionary to asn;
grant select on sys.v_$logmnr_logfile to asn;
grant select on sys.v_$logmnr_logs to asn;
grant select on sys.v_$logmnr_parameters to asn;
grant select on sys.v_$logmnr_session to asn;
grant select on sys.v_$logmnr_transaction to asn;
grant select on sys.v_$log to asn;
grant select on sys.v_$logfile to asn;
grant select on sys.v_$archived_log to asn;
```

And run it from the `sysdba` account. We found that we could create the user account from the system account, but that this account could not perform the `GRANT` statements:

```
SQL> @create_asn_account.txt
Grant succeeded.
Grant succeeded.
User altered.
Grant succeeded.
Grant succeeded.
Grant succeeded.
Grant succeeded.
Grant succeeded.
Grant succeeded.
Grant succeeded.
Grant succeeded.
Grant succeeded.
Grant succeeded.
Grant succeeded.
Grant succeeded.
Grant succeeded.
Grant succeeded.
```

4. Test the java environment for ASNCLP.

   The ASNCLP interface uses the Oracle JDBC drivers. To use this we need to set the `CLASSPATH` and `ORACLE_HOME`. We can do this either for a single session, or set it globally:

   ```
   > set CLASSPATH=C:\app\db2admin\product\11.1.0\db_1\oui\jlib\
   classes12.jar;%CLASSPATH%
   > set ORACLE_HOME=C:\oracle\product\10.2.0\db_1
   ```

   Now we can test the drivers as follows:

   ```
    >java  oracle.jdbc.driver.OracleDriver
   Exception in thread "main" java.lang.NoSuchMethodError: main
   ```

   This shows that the drivers are working.

   If the output is:

   ```
   Exception in thread "main" java.lang.NoClassDefFoundError: oracle/
   jdbc/driver/OracleDriver
   Caused by: java.lang.ClassNotFoundException: oracle.jdbc.driver.
   OracleDriver
   at java.net.URLClassLoader$1.run(Unknown Source)
   at java.security.AccessController.doPrivileged(Native Method)
   ```

```
at java.net.URLClassLoader.findClass(Unknown Source)
at java.lang.ClassLoader.loadClass(Unknown Source)
at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)
at java.lang.ClassLoader.loadClass(Unknown Source)
at java.lang.ClassLoader.loadClassInternal(Unknown Source)
```

Then the CLASSPATH and ORACLE_HOME variables are not set up correctly.

5. Create an ASNCLP Oracle source configuration file.

   When we invoke ASNCLP to work with an Oracle source we need to use the SET SERVER option to point to the Oracle database. So we need to create a parameter file for the Oracle source (c:\temp\asnclp_server_oracle.txt) and it will contain:

```
[ORCL]
type=ORACLE
Data source=ORCL
Host=4061AD4-L3ABK6B
port=1521
```

6. Create the replication control tables.

   We are now in a position to create the Q replication control tables. We can use the following ASNCLP command file:

```
ASNCLP SESSION SET TO Q REPLICATION;

SET SERVER CAPTURE TO CONFIG SERVER ORCL
FILE "c:\temp\asnclp_server_oracle.txt" ID system
 PASSWORD "passw0rd";

SET QMANAGER "QMA" FOR CAPTURE SCHEMA;
SET CAPTURE SCHEMA SOURCE ASN;

CREATE CONTROL TABLES FOR CAPTURE SERVER
USING RESTARTQ "CAPA.RESTARTQ"
ADMINQ "CAPA.ADMINQ" MEMORY LIMIT 64
MONITOR INTERVAL 600000;
```

   Run the file as:

```
> asnclp -f asnclp_create_cap_tables.txt
```

   If you get the error "ASN2022E The action ended in error. An SQL error was encountered. SQL message is "ORA-01017: invalid username/password; logon denied", it may mean that the password you specified was not in double quotes.

The ASNCLP command produces a file called `qreplcap.sql`, which contains the SQL to create the Q Capture control tables and insert a record into the `IBMQREP_CAPPARMS` table (all in Oracle). So we need to run this against the `orcl` Oracle database:

```
C:\oracle>sqlplus asn/asn

SQL> @qreplcap.sql

Table created.
Index created.
Table created.
Index created.
Table created.
Table created.
Table created.
Table created.
Table altered.
Table created.
Index created.
Table created.
Index created.
Table created.
Index created.
Table created.
Table created.
Table created.
Index created.
Table created.
Index created.
Table created.
Table created.
Table altered.
1 row created.
```

We can check the tables which were created using the following command:

```
SQL> SELECT table_name FROM user_tables;

TABLE_NAME
------------------------------
IBMQREP_CAPPARMS
IBMQREP_SENDQUEUES
```

```
IBMQREP_SUBS
IBMQREP_SRC_COLS
IBMQREP_SRCH_COND
IBMQREP_SIGNAL
IBMQREP_CAPTRACE
IBMQREP_CAPMON
IBMQREP_CAPQMON
IBMQREP_CAPENQ
IBMQREP_ADMINMSG
IBMQREP_IGNTRAN
IBMQREP_IGNTRANTRC
IBMQREP_CAPENVINFO
IBMQREP_EOLFLUSH
15 rows selected.
```

7. We need to switch on supplemental logging for each table that we want to replicate.

   Create and populate a source table:

```
> sqlplus system/passw0rd@orcl

SQL> CREATE TABLE scott.tb1 (c1 number(5), c2 number(5), c3
number(5)) ;

SQL> CREATE UNIQUE INDEX scott.tb1I on scott.tb1 (c1 asc) ;
```

   Check the table definition as follows:

```
SQL> desc scott.tb1 ;

 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 C1                                                 NUMBER(5)
 C2                                                 NUMBER(5)
 C3                                                 NUMBER(5)
```

   Populate the source table with two rows:

```
SQL> INSERT INTO scott.tb1 values (1,1,1);
SQL> INSERT INTO scott.tb1 values (2,2,2);
```

   And switch on supplemental logging for the table:

```
SQL> ALTER TABLE scott.tb1 ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS
;

Table altered.
```

We have finished all the work we need to do on the Oracle server, we now turn our attention to the DB2 target server.

Create the Q Apply control tables on The Q Apply control tables should be created as in the *Unidirectional direction – The Q replication layer – Creating Q Apply control tables on DB2B* section.

8. Create a Replication Queue Map.

   We need to create a Replication Queue Map going from `QMA` on the Oracle server to `QMB` on the DB2 target server. We do this using the following ASNCLP commands:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO CONFIG SERVER ORCL
FILE "c:\temp\asnclp_server_oracle.txt" ID system
 PASSWORD "passw0rd";

SET SERVER TARGET  TO DB DB2B ;
SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY   SCHEMA        ASN;

SET QMANAGER QMA FOR CAPTURE SCHEMA;
SET QMANAGER QMB FOR APPLY   SCHEMA;

CREATE REPLQMAP "RQMO2B"
USING
ADMINQ "CAPA.ADMINQ.REMOTE"
RECVQ  "CAPA.TO.APPB.RECVQ"
SENDQ  "CAPA.TO.APPB.SENDQ.REMOTE"
NUM APPLY AGENTS 3
MEMORY LIMIT 9
ERROR ACTION S
HEARTBEAT INTERVAL 0
MAX MESSAGE SIZE 4;
```

   Run the preceding file as:

```
> asnclp -f SYSA_crt_rqmo2b.asnclp
```

9. Create a Q subscription for the Oracle source.

   When we create the Q subscription, we have two options for dealing with the initial load. We can either specify that we should not do an initial load (`HAS LOAD PHASE = N`), or specify that Q replication should choose the best option (`HAS LOAD PHASE = I`, which is the default, and `LOAD TYPE = 0`).

In this first example, we have specified that there will not be an initial load:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO CONFIG SERVER ORCL
FILE "c:\temp\asnclp_server_oracle.txt" ID system
 PASSWORD "passw0rd";

SET QMANAGER "QMA" FOR CAPTURE SCHEMA;
SET CAPTURE SCHEMA SOURCE ASN;

SET SERVER TARGET TO DB DB2B;
SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY SCHEMA ASN;
SET QMANAGER QMA FOR CAPTURE SCHEMA;
SET QMANAGER QMB FOR APPLY SCHEMA;
CREATE QSUB
USING REPLQMAP RQMO2B
(SUBNAME QSUBTB2  SCOTT.TB1
  OPTIONS HAS LOAD PHASE N
  TARGET NAME DB2ADMIN.TGTTB1
  TYPE USERTABLE
  KEYS(C1) );
```

Run the above file as:

```
> asnclp -f SYSA_qsub_o2b_Noload.asnclp
```

We have now set up the Q replication environment and can move on to starting Q Capture and Q Apply. As with all the scenarios, we need to start the Listeners and Channels on QMA and QMB.

10. Start Q Capture.

    We need to start Q Capture on the server that contains the Oracle database and the QMA Queue Manager:

    ```
    > asnoqcap capture_server=orcl
    ```

11. Start Q Apply.

    We need to start Q Apply on the server that contains the DB2B DB2 database and the QMB Queue Manager:

    ```
    > start asnqapp apply_server=DB2B   APPLY_PATH="C:\TEMP"
    ```

12. Test replication.

    We are now in a position to test replication. We can insert a record into the Oracle database using the following commands:

```
SQL> INSERT INTO scott.tb1 values (3,3,3);

1 row created.

SQL> commit ;

Commit complete.
```

    And let's check on DB2B that the record has been replicated:

```
> db2 connect to db2b

> db2 select * from db2admin.tgttb1

C1          C2          C3
----------- ----------- -----------
          3           3           3
  1 record(s) selected.
```

    We can see that the record has been replicated (and that no initial load was performed). Note that as we did not perform an initial load, we did not have to set up any federation objects on DB2B.

    Let's move on to the second example, where we have requested an initial load. If we want to perform an initial load of the target DB2 database, then we need to perform some additional steps to the ones detailed previously. The first step is to enable the target instance for federation.

    For the instance on which DB2B resides, issue:

```
> db2 update dbm cfg using FEDERATED  YES
```

    We should then stop and start the instance for the change to take effect.

    On the target database (DB2B), we need to create a:

    - Oracle wrapper definition
    - Oracle server definition
    - User mapping

    Create an Oracle wrapper using the following command:

```
> db2 CREATE WRAPPER NET8 LIBRARY 'db2net8.dll'
```

    Create a server definition using the following command:

```
> db2 CREATE SERVER SORA TYPE ORACLE VERSION '10g' WRAPPER NET8
OPTIONS( ADD NODE 'orcl')
```

Create a User Mapping using the following command:

```
> db2 CREATE USER MAPPING FOR GB043449 SERVER SORA OPTIONS ( ADD
REMOTE_AUTHID 'scott', ADD  REMOTE_PASSWORD 'passw0rd')
```

Note that the password is the password for scott on the Oracle database (`passw0rd` in our test).

The following Q subscription example has Q Apply choosing the best load method, and uses a nickname from `DB2B` to the Oracle database to perform the initial load:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;

SET SERVER CAPTURE TO CONFIG SERVER ORCL
FILE "c:\temp\asnclp_server_oracle.txt" ID system
 PASSWORD "passw0rd";

SET QMANAGER "QMA" FOR CAPTURE SCHEMA;
SET CAPTURE SCHEMA SOURCE ASN;

SET SERVER TARGET TO DB DB2B;
SET CAPTURE SCHEMA ASN;
SET APPLY SCHEMA ASN;
SET QMANAGER QMA FOR CAPTURE SCHEMA;
SET QMANAGER QMB FOR APPLY SCHEMA;
CREATE QSUB (SUBNAME QSUBTB3
  REPLQMAP RQMO2B SCOTT.TB1
  OPTIONS HAS LOAD PHASE I TARGET NAME GB043449.TGTTB3
  TYPE NEW NICKNAME RMT SERVERNAME DB2B
  NICKTB1 LOAD TYPE 1 );
```

Note that we are creating a nickname in the `DB2B` with a schema of `GB043449`. This is the schema that we specified when we created the user mapping in a previous step.

Before we can start Q Capture and Q Apply, we need to create a password file for the Oracle data source on the server on which `DB2B` resides:

```
C:\temp>asnpwd init encrypt password

ASN1981I  "Asnpwd" : "" : "Initial". The program completed
successfully using password file "asnpwd.aut".

C:\temp>asnpwd add alias orcl id system password passw0rd

ASN1981I  "Asnpwd" : "" : "Initial". The program completed
successfully using password file "asnpwd.aut".
```

```
C:\temp>asnpwd list
Alias: ORCL  ID: system
Number of Entries: 1
```

We can now start Q Capture and Q Apply and test replication as the preceding commands show. Note that as we specified that Q Apply should perform an initial load then after we insert one record into the Oracle table, we will see three records in DB2B, the two records from the initial load and the record inserted into the Oracle table.

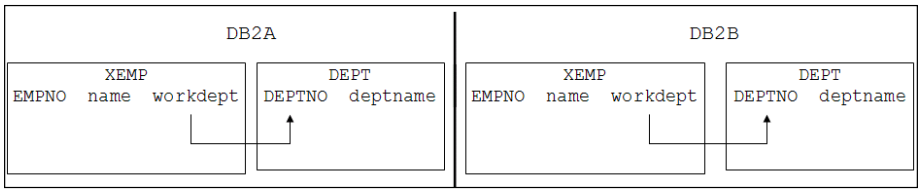
Conflict detection examples

We will work through two examples of conflict detection—one involving a foreign key relationship, and the second example involving the update/delete of the same record on different servers.

Conflict detection—foreign keys

Let's work through an example of conflict detection when foreign keys are involved using a cascaded delete and employing bidirectional replication.

We have two servers DB2A and DB2B with tables XEMP and DEPT on each. Table DEPT has a foreign key on Table XEMP:



In our test, we delete a row from table XEMP on server DB2A (which does a cascade delete all in DEPT on DB2B). At the same time, we insert a row into table DEPT on DB2B based on the foreign key value in XEMP.

We will use three CLP sessions called CLP-A, CLP-B, and CLP-C.

Let's create the two test tables XEMP and DEPT:

From CLP-A, issue:

```
$ db2 "CREATE TABLE db2admin.xemp (empno CHAR(6) NOT NULL, name
VARCHAR(12) NOT NULL, workdept CHAR(3) )"
```

```
$ db2 "ALTER TABLE db2admin.xemp ADD CONSTRAINT xemp_prim PRIMARY KEY
(empno)"
$ db2 "CREATE TABLE db2admin.dept (deptno CHAR(3) NOT NULL, deptname
CHAR(10) NOT NULL)"
$ db2 "ALTER TABLE db2admin.dept ADD CONSTRAINT dept_prim PRIMARY KEY
(deptno)"
$ db2 "ALTER TABLE db2admin.xemp FOREIGN KEY RDE (workdept) REFERENCES
db2admin.dept ON DELETE CASCADE"
```

We cannot enter a record into XEMP with a WORKDEPT if there isn't a corresponding column in the DEPT table for WORKDEPT.

Let's first insert records into DEPT:

```
$ db2 "insert into dept values ('A01','Ops') "
$ db2 "insert into dept values ('A02','Serv') "
$ db2 "insert into dept values ('A03','Maint') "
```

Now populate the XEMP table:

```
$ db2 "insert into xemp values ('000001','Sue','A01') "
$ db2 "insert into xemp values ('000002','Mary','A01') "
$ db2 "insert into xemp values ('000003','Helen','A02') "
```

So let's try to insert a row into XEMP with a WORKDEPT value that is not in DEPT.

From CLP-A, issue:

```
$ db2 "insert into xemp values ('000004','Helen','A04') "
SQL0530N  The insert or update value of the FOREIGN KEY "DB2ADMIN.
XEMP.RDE" is not equal to any value of the parent key of the parent
table.  SQLSTATE=23503
```

And let's test that the cascaded delete works. We will delete from DEPT and check that the delete has cascaded to EMP:

```
$ db2 "select * from xemp"

EMPNO  NAME         WORKDEPT
------ ------------ --------
000001 Sue          A01
000002 Mary         A01
000003 Helen        A02

$ db2 "delete from dept where deptno = 'A01' "
```

```
$ db2 "select * from xemp"

EMPNO  NAME         WORKDEPT
------ ------------ --------
000003 Helen        A02
```

We can see that the cascaded delete has worked.

So let's put back the two records we deleted.

```
$ db2 "insert into xemp values ('000001','Sue','A01') "
$ db2 "insert into xemp values ('000002','Mary','A01') "

$ db2 "select * from xemp"

EMPNO  NAME         WORKDEPT
------ ------------ --------
000001 Sue          A01
000002 Mary         A01
000003 Helen        A02
  3 record(s) selected.
```

Set up bidirectional replication for tables `XEMP` and `DEPT` between `DB2A` and `DB2B` (Refer to the *Bidirectional replication* section).

The Q subscription load file is called `SYSA_loadfk.asnclp` and the content file is called `SYSA_contfk.txt`:

```
set subgroup "TABT2";

set server multidir to db "DB2A";
set server multidir to db "DB2B";

set multidir schema "DB2A".ASN;
set multidir schema "DB2B".ASN;

set connection SOURCE "DB2A".ASN
TARGET "DB2B".ASN replqmap "RQMA2B";

set connection SOURCE "DB2B".ASN
TARGET "DB2A".ASN replqmap "RQMB2A";

set tables(DB2A.ASN.DB2ADMIN.DEPT, DB2B.ASN.DB2ADMIN.DEPT);

CREATE QSUB subtype b
from node db2a.asn
SOURCE
all changed rows y

has load phase I
```

```
TARGET
conflict rule A
conflict action I
error action s
load type 0
oksqlstates "000"

from node db2b.asn
SOURCE
all changed rows N
has load phase N
TARGET
conflict rule A
conflict action F
error action s
load type 0
oksqlstates "000";

set tables (DB2A.ASN.DB2ADMIN.XEMP, DB2B.ASN.DB2ADMIN.XEMP);

CREATE QSUB subtype b
from node db2a.asn
SOURCE
all changed rows y
has load phase I
TARGET
conflict rule A
conflict action I
error action s
load type 0
oksqlstates "000"

from node db2b.asn
SOURCE
all changed rows N
has load phase N
TARGET
conflict rule A
conflict action F
error action s
load type 0
oksqlstates "000";
```

From CLP-A, issue:

```
$ asnclp -f SYSA_loadfk.asnclp
```

Use the query in the *Q subscription maintenance – To list the attributes of a Q subscription* section of *Chapter 1, Administration Tasks*, to list out the attributes of the Q subscription.

The result from CLP-A is:

```
SUBNAME    SUB_ID     C B H L S STATE_TIME           SUBGP      SN TN
---------- ---------- - - - - - -------------------------- -------
DEPT0001         - Y Y N I N 2010-01-20-20.23.35.515000 TABT2    1  2
XEMP0001         - Y Y N I N 2010-01-20-20.23.35.562000 TABT2    1  2

SUBNAME    TRGNAME    TT APPSCHEMA  SENDQ
---------- ---------- -- ---------- -------------------------
DEPT0001   DEPT       1  ASN        CAPA.TO.APPB.SENDQ.REMOTE
XEMP0001   XEMP       1  ASN        CAPA.TO.APPB.SENDQ.REMOTE

SUBNAME    SUB_ID     C B H L S STATE_TIME           SUBGP      SN TN
---------- ---------- - - - - - -------------------------- -------
DEPT0001         - Y Y N I N 2010-01-20-20.23.35.515000 TABT2    1  2
XEMP0001         - Y Y N I N 2010-01-20-20.23.35.562000 TABT2    1  2
```

The result from CLP-B is:

```
SUBNAME    SRCOWNER   SRCNAME    TRGSRV     TRGALIAS TRGOWNER
---------- ---------- ---------- ---------- -------- ----------
DEPT0002   DB2ADMIN   DEPT       DB2A       DB2A     DB2ADMIN
XEMP0002   DB2ADMIN   XEMP       DB2A       DB2A     DB2ADMIN

SUBNAME    TRGNAME    TT APPSCHEMA  SENDQ
---------- ---------- -- ---------- -------------------------
DEPT0002   DEPT       1  ASN        CAPB.TO.APPA.SENDQ.REMOTE
XEMP0002   XEMP       1  ASN        CAPB.TO.APPA.SENDQ.REMOTE

SUBNAME    SUB_ID      C B H L S STATE_TIME             SUBGP
SN TN
---------- ---------- - - - - - -------------------------- -------
DEPT0002         - N Y N N I 2010-01-20-20.23.35.453000 TABT2    2  1
XEMP0002         - N Y N N I 2010-01-20-20.23.35.484000 TABT2    2  1
```

Start Q Capture—follow the instructions in the *Q Capture administration – Starting Q Capture* section of *Chapter 6*.

Start Q Apply—follow the instructions in the *Q Apply administration – Starting Q Apply* section of *Chapter 6*.

So now let's issue a delete on `DB2A` from `DEPT` where `DEPTNO` is `A01` and insert a row on `DB2B` into `XEMP` with a `WORKDEPT` value of `A01` at the same time. We will do this using a script.

Create a script file called `c:\temp\fk00.bat`. This will contain:

```
db2cmd fk01
db2cmd fk02
```

Create the `.BAT` file `fk01.bat` to contain:

```
@echo fk01 - delete
db2 connect to DB2A user
db2 select current timestamp from sysibm.sysdummy1 with ur
db2 delete from dept where deptno = 'A01'
db2 select current timestamp from sysibm.sysdummy1 with ur
```

Create the `.BAT` file `fk02.bat` to contain:

```
@echo fk02 - insert
db2 connect to DB2B
db2 select current timestamp from sysibm.sysdummy1 with ur
db2 insert into xemp values ('00004','Heather','A01')
db2 select current timestamp from sysibm.sysdummy1 with ur
```

From CLP-C, run fk00 as follows:

```
C:\temp> fk00
```

When we run the `fk00.bat` file, two command windows open

fk00	
fk01	**fk02**
fk01 - delete	fk02 - insert
> db2 connect to DB2A	**> db2 connect to DB2B**
> db2 select current timestamp from sysibm.sysdummy1 with ur	**> db2 select current timestamp from sysibm.sysdummy1 with ur**
1 --- 2010-01-20-21.08.05.968000	1 --- 2010-01-20-21.08.06.093000
> db2 delete from dept where deptno = 'A01' DB20000I The SQL command completed successfully.	**> db2 insert into xemp values ('00004','Anita','A01')** DB20000I The SQL command completed successfully.
> db2 select current timestamp from sysibm.sysdummy1 with ur	**> db2 select current timestamp from sysibm.sysdummy1 with ur**
1 --- 2010-01-20-21.08.06.468000	1 --- 2010-01-20-21.08.06.484000

We are trying to propagate a delete from DB2A to DB2B and an insert from DB2B to DB2A.

Check whether the DELETE and INSERT have worked as per the preceding output.

From CLP-A issue:

```
$ db2 connect to db2a

$ db2 select * from dept

DEPTNO DEPTNAME
------ ----------
A02    Serv
A03    Maint

  2 record(s) selected.

$ db2 select * from xemp

EMPNO  NAME         WORKDEPT
------ ------------ --------
000003 Helen        A02

  1 record(s) selected.
```

From CLP-B issue:

```
$ db2 connect to db2b

$ db2 select * from dept

DEPTNO DEPTNAME
------ ----------
A02    Serv
A03    Maint

  2 record(s) selected.

$ db2 select * from xemp

EMPNO  NAME         WORKDEPT
------ ------------ --------
000003 Helen        A02

  1 record(s) selected.
```

We can see that the INSERT issued on DB2B has not been propagated but the DELETE issued on DB2A has been propagated.

Let's check the IBMQREP_TARGETS table on each system. Use the following query:

```
$ db2 "SELECT SUBSTR(subname,1,10) AS subname, conflict_action FROM asn.
ibmqrep_targets"
```

The result from CLP-A is:

```
SUBNAME    CONFLICT_ACTION
---------- ---------------
DEPT0002   F
XEMP0002   F
```

The result from CLP-B is:

```
SUBNAME    CONFLICT_ACTION
---------- ---------------
DEPT0001   I
XEMP0001   I
```

The CONFLICT_ACTION for Q subscriptions DEPT0002/XEMP0002 is F—which means that Q Apply on DB2B will force the changes that it receives from DB2A.

The CONFLICT_ACTION for Q subscriptions DEPT0001/XEMP0001 is I—which means that Q Apply on DB2A will not force the change that it receives from DB2B.

Let's remind ourselves of what we are trying to do—we are trying to propagate a DELETE from DB2A to DB2B and an INSERT from DB2B to DB2A.

Therefore the DELETE from DB2A will be forced onto DB2B and the INSERT from DB2B to DB2A will be ignored and sent to the IBMQREP_EXCEPTIONS table on DB2A.

Select from the IBMQREP_EXCEPTIONS table on both systems using the query:

```
$ db2 "select exception_time, substr(subname,1,10) as subname,reason,
sqlcode, sqlstate, substr(operation,1,10) as op, is_applied as a,
conflict_rule,src_trans_time from asn.ibmqrep_exceptions"
```

From CLP-A, running the preceding query results in:

```
EXCEPTION_TIME             SUBNAME    REASON       SQLCODE
SQLSTATE OP         A CONFLICT_RULE SRC_TRANS_TIME
-------------------------- ---------- ------------ ----------- ------
-- ---------- - ------------ --------------------------
2010-01-20-21.08.07.828000 XEMP0002   SQLERROR            -530 23503
INSERT     N A             2010-01-20-21.08.06.000024
```

Let's look at the contents of the IBMQREP_EXCEPTIONS table in more detail. We can see what caused the exception by looking in the TEXT column. The query we will use is:

```
$ db2 "select is_applied, substr(text,1,1000) from asn.ibmqrep_
exceptions"
```

On CLP-A:

```
N           INSERT INTO "DB2ADMIN"."XEMP"( "EMPNO", "NAME",
"WORKDEPT") VALUES ('00004 ', 'Anita', 'A01')
```

On CLP-B, the IBMQREP_EXCEPTION table is empty.

And let's check the status of Q Capture and Q Apply. Q Apply on DB2A has terminated, and its log shows the following message:

```
2010-01-20-21.08.07.765000 <appAgnt::handleRowError> ASN8999D A SQL
error occured for subscription "XEMP0002" for receive queue "CAPB.
TO.APPA.RECVQ" (SQL code "-530"), while applying a "INSERT

2010-01-20-21.08.07.765000 <appAgnt::handleRowError> ASN0552E  "Q
Apply" : "ASN" : "BR00000AG005" : The program encountered an SQL
error. The server name is "". The SQL request is "INSERT". The table
name is ""DB2ADMIN"."XEMP"". The SQLCODE is "-530". The SQLSTATE is
"23503". The SQLERRMC is "DB2ADMIN.XEMP.RDE". The SQLERRP is " ".

2010-01-20-21.08.07.843000 <appAgnt::handleRowError> ASN7522E  "Q
Apply" : "ASN" : "BR00000AG005" : The Q Apply program stopped because
it encountered an error for Q subscription "XEMP0002" (receive queue
"CAPB.TO.APPA.RECVQ", replication queue map "RQMB2A").

2010-01-20-21.08.07.859000 <QAtxs::applyTran> ASN0589I  "Q Apply" :
"ASN" : "BR00000AG005" The program received an unexpected return code
"2012" from routine "handleRowError".

2010-01-20-21.08.07.859000 <QAtxs::applyTran> ASN8999D problem
applying transaction because of a row error

2010-01-20-21.08.07.859000 <appAgntMain-retry> ASN0589I  "Q Apply" :
"ASN" : "BR00000AG005" The program received an unexpected return code
"2012" from routine "applyTran".

2010-01-20-21.08.07.859000 <appAgntMain> ASN8999D Apply agent
'BR00000AG005' terminating with code 2012
```

Because we had the ERROR ACTION set to S, this means that Q Apply will stop without applying the transaction. Perhaps a better value would be the default value of Q, which means that Q Apply stops reading from the Receive Queue and all conflicting rows are inserted into the IBMQREP_EXCEPTIONS table.

From the Q Apply log and the entry in the IBMQREP_EXCEPTIONS table, we can see that Q Apply is trying to insert a row into the XEMP table, that this is failing with an SQLSTATE of 23503 and that the Q subscription name is XEMP0002.

The row that Q Apply is trying to apply on DB2A is still on the Receive Queue. We cannot simply delete the message, because of the dense numbering safeguard.

What we have to do is tell Q Apply to ignore the SQL that the INSERT statement is generating. We do this by inserting the SQLSTATE into the OKSQLSTATES column of the IBMQREP_TARGETS table on DB2A.

Let's first see what is in the table.

```
$ db2 "select substr(subname,1,10) as subname, oksqlstates from asn.
ibmqrep_targets "

SUBNAME    OKSQLSTATES
---------- -----------
DEPT0002   000
XEMP0002   000
```

So we need to update the `OKSQLSTATES` value for Q subscription `XEMP0002`.

We have the `SQLSTATE` that we want to add to the `IBMQREP_EXCEPTIONS` table, and can code an `UPDATE` statement as follows:

```
$ db2 "UPDATE asn.ibmqrep_targets SET OKSQLSTATES = ' "000",""23503"" '
WHERE SUBNAME = 'XEMP0002' "
```

Note the use of the two sets of double quotation marks around the `SQLSTATE` value `23503`.

And selecting from the table:

```
SUBNAME    OKSQLSTATES
---------- ------------------------
DEPT0002   000
XEMP0002    000,"23503"
```

Now we can try and start Q Apply on `DB2A`. This time it will start.

To test that everything is back to normal, insert a record on `DB2B` and check that it has been replicated to `DB2A`.

From CLP-B, issue:

```
$ db2 "INSERT INTO xemp(empno, name, workdept) VALUES('00004', 'Anna',
'A02')"
```

From CLP-A, issue:

```
$ db2 select * from xemp

EMPNO  NAME         WORKDEPT
------ ------------ --------
00004  Anna         A02
000003 Helen        A02
```

The last task we have to do is to remove the `23503 SQLTSTATE` from the `IBMQREP_TARGETS` table:

```
$ db2 "UPDATE asn.ibmqrep_targets SET OKSQLSTATES = ' "000" ' WHERE
SUBNAME = 'XEMP0002' "
```

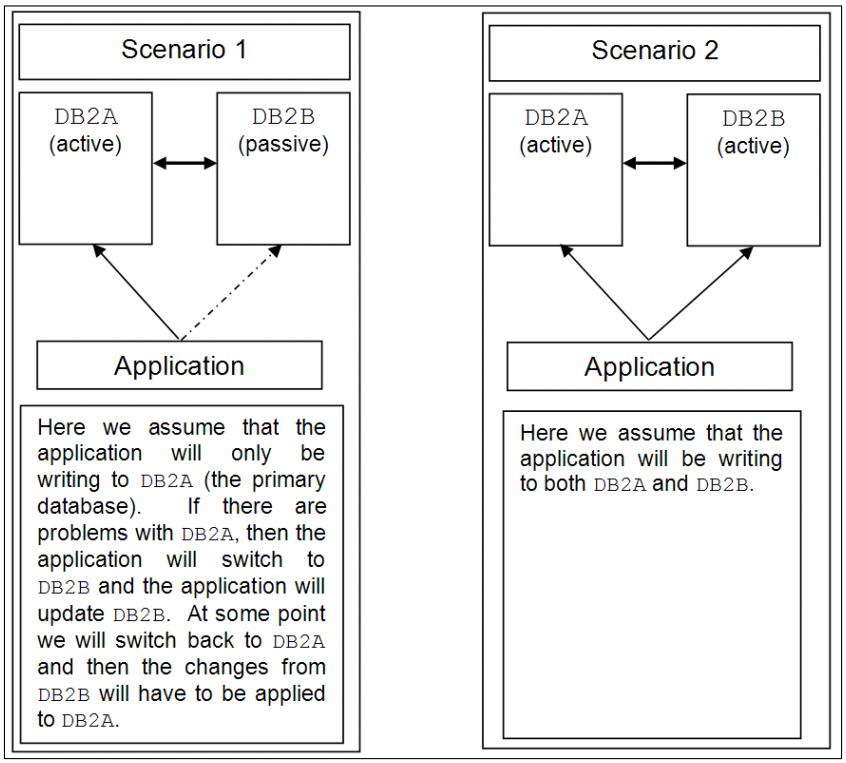
We have now finished looking at conflict detection with foreign keys.

Conflict detection—update/delete conflict

The example in this section looks at `UPDATE/DELETE` conflict detection and resolution.

We have two servers `DB2A` and `DB2B` with table `HMTAB` on each. We are using bidirectional replication. On server `DB2A`, we delete a row from table `HMTAB` and on server `DB2B`, we update the same row at exactly the same time.

There are two main scenarios using bidirectional replication and are shown in the following diagram, where we normally update only one side (**Scenario 1**) or where we update both sides (**Scenario 2**).



The questions we need answers to are:

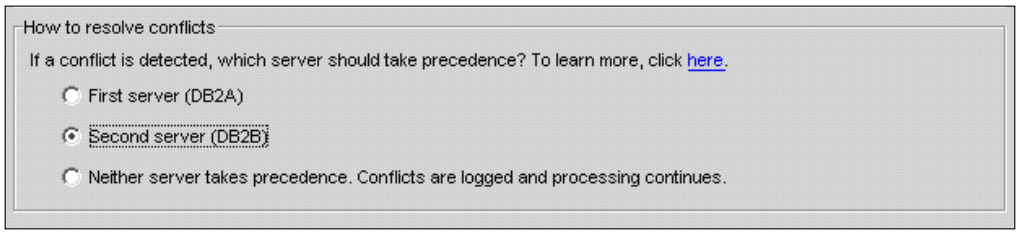
- Is this scenario picked up by conflict detection?
- Which server wins?

Note that the DB2 Information Center says …*Even the highest level of value-based conflict detection (checking all columns) in bidirectional replication is not as robust as version-based conflict detection in* P2P *replication. Some situations might result in a conflict not being detected. Recommendation: If you expect conflicts by application design, then choose a* P2P *configuration*.

Scenario 1—Bidirectional active/passive

In Scenario 1, we assume that the application is only writing to one database at a time. Typically, applications will be updating `DB2A` and not `DB2B`, and `DB2B` will be acting as a read-only copy of the data. If there is a problem with `DB2A` and it goes down, then the application will switch to the other server and start updating `DB2B`. After we fix the problem on `DB2A` and bring it back up, we will want to send the changes from `DB2B` back over to `DB2A`, and we will want to force any conflicts because `DB2B` will have the most recent information. We should not end up in a situation where `DB2A` has to force conflicts on to `DB2B` because no applications will update `DB2B` unless there is a problem with `DB2A`.

If we have two servers `DB2A` and `DB2B` and we want to make `DB2A` the primary database (the one which will normally be used for updates), then select `DB2B` as the server which should take precedence if a conflict is detected on screen 6 "Conflicts" of Create Q Subscriptions (as shown in the following screenshot):



The answer to which sever wins depends on the `CONFLICT_ACTION` column in `IBMQREP_TARGETS` (`I` is the default). This value determines the action to take when a row change is invalid. For example, row to delete/update not found.

So let's set up a scenario to test out the delete/update conflict detection.

To test out Scenario 1, we will use three CLP sessions—CLP-A, CLP-B, and CLP-C.

In general, we will follow the bidirectional setup steps in the *Bidirectional replication* section, but with some additional steps in each layer.

1. Perform the DB2 layer step and in addition perform the following, where we will create our test table called `HMTAB`:

   From CLP-A, issue:

   ```
   $ db2 "create table eric.hmtab (id int not null, name char(10),
   dept char(5)) "
   $ db2 "alter table eric.hmtab add primary key (id) "
   $ db2 "insert into eric.hmtab values(1,'Heather','Ops') "
   ```

2. Perform the WebSphere MQ layer—there are no addition steps.
3. For the Q replication layer:

   Create the two Replication Queue Maps `RQMA2B` and `RQMB2A` as described in the *Bidirectional replication – The Q replication layer – Creating a Replication Queue Map for DB2A to DB2B* and *Creating a Replication Queue Map for DB2B to DB2A* sections.

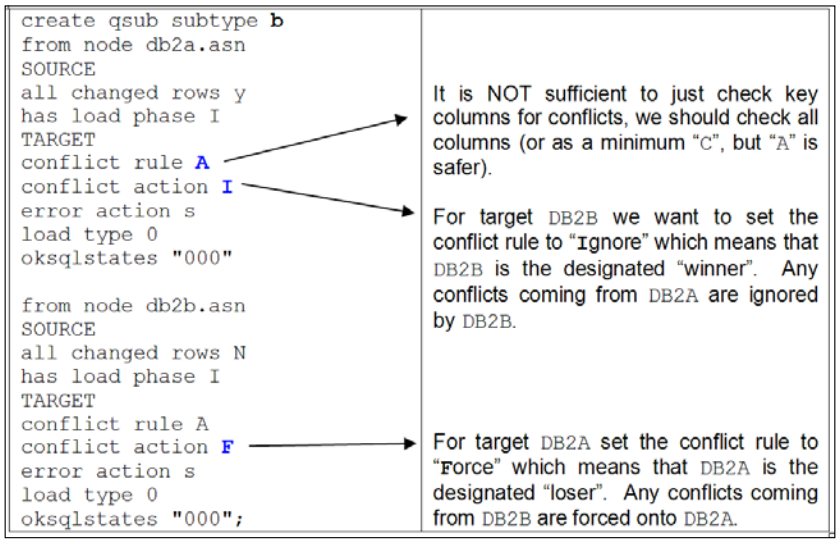
Now we come to defining the Q subscription. What should we specify for the conflict rules? Say we want `DB2A` to be the primary database and `DB2B` to be the backup database (which means that `DB2B` is the winner and `DB2A` the loser in any conflict).

The header section of the content file looks like this:

```
set subgroup "TABT2";

set server multidir to db "DB2A";
set server multidir to db "DB2B";

set multidir schema "DB2A".ASN;
set multidir schema "DB2B".ASN;

set connection SOURCE "DB2A".ASN
TARGET "DB2B".ASN replqmap "RQMA2B";

set connection SOURCE "DB2B".ASN
TARGET "DB2A".ASN replqmap "RQMB2A";

set tables("DB2A".ASN.ERIC.HMTAB, "DB2B".ASN.FRED.HMTAB);
```

Next is the option section of the content file. This tells Q replication how to handle conflicts and so on. The section is divided into two sub-sections, each starting with the keyword `from node`. The first sub-section says `from node db2a.asn`, so we are coming from `DB2A`. Half way down the section it says `TARGET` and then we specify `conflict rule (A)`, `conflict action (I)`, `error action (s)`, `load type (0)`, and `oksqlstates ("000")`. This means that at the target side (`DB2B`), we want a conflict action of `I`, which means that any conflicts coming from `DB2A` are ignored.

The Q subscription options section will look like this:



Let's store the Q subscription code in a content file called `SYSA_contdelupd.txt`, with a load file called `SYSA_loaddelupd.asnclp`.

Create the Q subscription. From CLP-A, issue:

```
$ asnclp -f SYSA_loaddelupd.asnclp
```

Use the query in the *Q subscription maintenance – To list the attributes of a Q subscription* section of *Chapter 6*, to list out the attributes of the Q subscription.

The result from CLP-A is:

```
SUBNAME     SRCOWNER    SRCNAME     TRGSRV     TRGALIAS TRGOWNER . . . .>
HMTAB0001   ERIC        HMTAB       DB2B       DB2B     FRED       . . . .>

TRGNAME     TT APPSCHEMA   SENDQ                          . . . .>
HMTAB       1  ASN         CAPA.TO.APPB.SENDQ.REMOTE      . . . .>
```

```
SUB_ID      C B H L S STATE_TIME                 SUBGP      SN TN
----- - - - - - -------------------------- ---------- -- -- -- --
          - Y Y N I N 2007-04-08-16.32.50.531000 TABT2      1  2
```

The result from CLP-B is:

```
SUBNAME    SRCOWNER   SRCNAME    TRGSRV    TRGALIAS TRGOWNER . . . .>
HMTAB0002  FRED       HMTAB      DB2A       DB2A    ERIC     . . . .>

TRGNAME    TT APPSCHEMA  SENDQ                       . . . .>
HMTAB      1  ASN        CAPB.TO.APPA.SENDQ.REMOTE   . . . .>

SUB_ID      C B H L S STATE_TIME                 SUBGP      SN TN
----------- - - - - - -------------------------- ---------- -- --
          - N Y N I I 2007-04-08-16.32.50.546000 TABT2      2  1
```

To start Q Capture, follow the instructions in the *Q Capture administration – To start Q Capture* section of *Chapter 6*.

To start Q Apply, follow the instructions in the *Q Apply administration – To start Q Apply* section of *Chapter 6*.

We can now insert one row into the `HMTAB` table on `DB2A`. From CLP-A, issue:

```
$ db2 "insert into eric.hmtab values(2,'Helen','Trn') "
```

Check that the row has been replicated to `DB2B`.

From CLP-B, issue:

```
$ db2 "select * from fred.hmtab"

ID          NAME       DEPT
----------- ---------- -----
          1 Heather    Ops
          2 Helen      Trn
```

We should see the row we inserted with an `ID` value of `2`.

Now insert one row into the `HMTAB` table on `DB2B`. From CLP-B, issue:

```
$ db2 "insert into fred.hmtab values(3,'Chantal','Mgm') "
```

Check that the row has been replicated to `DB2A`. From CLP-A, issue:

```
$ db2 "select * from eric.hmtab"
```

```
ID          NAME       DEPT
----------- ---------- -----
          1 Heather    Ops
          2 Helen      Trn
          3 Chantal    Mgm
```

We should see the row we inserted with an `ID` value of `3`.

We have inserted a row on `DB2A` and seen it replicate to `DB2B` and we have inserted a row into `DB2B` and seen it replicate to `DB2A`. We have now shown that bidirectional replication is working.

We can now run a script to delete record two from table `HMTAB` in database `DB2A` and simultaneously update the same record in database `DB2B`. We do this as follows.

Create a `BAT` file called `c:\temp\conf00.bat`. This will contain:

```
db2cmd conf01
db2cmd conf02
```

Create the `BAT` file `conf01.bat` to contain:

```
@echo conf01 - delete
db2 connect to DB2A
db2 select current timestamp from sysibm.sysdummy1 with ur
db2 delete from eric.hmtab where id = 2
db2 select current timestamp from sysibm.sysdummy1 with ur
```

Create the `BAT` file `conf02.bat` to contain:

```
@echo conf02 - update
db2 connect to DB2B
db2 select current timestamp from sysibm.sysdummy1 with ur
db2 update fred.hmtab set dept = 'XXX' where id = 2
db2 select current timestamp from sysibm.sysdummy1 with ur
```

From CLP-C run `conf00` as follows:

```
C:\temp> conf00
```

When we run the `conf00.bat` file, two command windows open:

conf00	
conf01	**conf02**
conf01 - delete	conf02 - update
$ db2 connect to DB2A	**$ db2 connect to DB2B**
$ db2 select current timestamp from sysibm.sysdummy1 with ur	**$ db2 select current timestamp from sysibm.sysdummy1 with ur**
1 --- 2007-04-08-16.54.28.609000	1 --- 2007-04-08-16.54.28.312000
$ db2 delete from eric.hmtab where id = 2	**$ db2 update fred.hmtab set dept = 'XXX' where id = 2**
DB20000I The SQL command completed successfully.	DB20000I The SQL command completed successfully.
$ db2 select current timestamp from sysibm.sysdummy1 with ur	**$ db2 select current timestamp from sysibm.sysdummy1 with ur**
1 --- 2007-04-08-16.54.28.796000	1 --- 2007-04-08-16.54.28.500000

Check that the `DELETE` and `UPDATE` have worked as per the preceding output.

Now select from the table from both databases. From CLP-A, issue:

```
$ db2 "select * from eric.hmtab"

ID          NAME       DEPT
----------- ---------- -----
          1 Heather    Ops
          2 Helen      XXX
          3 Chantal    Mgm
```

From CLP-B, issue:

```
$ db2 "select * from fred.hmtab"
ID          NAME       DEPT
----------- ---------- -----
          1 Heather    Ops
          2 Helen      XXX
          3 Chantal    Mgm
```

We can see that the `DELETE` has been processed for record `2`—which means that the `UPDATE` we issued failed.

Let's find out why. First let us check the `IBMQREP_TARGETS` table on each system using the following query:

```
$ db2 "select substr(subname,1,10) as subname, conflict_action from asn.
ibmqrep_targets"
```

From CLP-A, running the preceding query results in:

```
SUBNAME    CONFLICT_ACTION
---------- ---------------
HMTAB0002  F
```

On `DB2A`, we can see that the `CONFLICT_ACTION` value for Q subscription `HMTAB0002` is `F`.

From CLP-B, running the preceding query results in:

```
SUBNAME    CONFLICT_ACTION
---------- ---------------
HMTAB0001  I
```

On `DB2B`, we can see that the `CONFLICT_ACTION` value for Q subscription `HMTAB0001` is `I`.

Let's remind ourselves what `I` and `F` mean:

- `I`: This is a default value. Q Apply does not apply the conflicting row but applies other rows in the transaction.
- `F`: Q Apply tries to force the change.

Next let's select from the `IBMQREP_EXCEPTIONS` table on both `DB2A` and `DB2B`.

The query we will use is:

```
$ db2 "select exception_time, substr(subname,1,10) as subname, reason,
sqlcode, sqlstate, substr(operation,1,10) as op, is_applied as a,
conflict_rule as CR, src_trans_time from asn.ibmqrep_exceptions"
```

From CLP-A, running the preceding query results in:

```
EXCEPTION_TIME          SUBNAME    REASON    SQLCODE   SQLSTATE OP  >
2007-04-08-16.54.31.796000 HMTAB0002  NOTFOUND  100 02000 UPDATE    >

A CR SRC_TRANS_TIME
Y A  2007-04-08-15.54.28.000036
```

What the preceding command tells us is that Q subscription HMTAB0002 tried to UPDATE a record (on DB2A), but the record was NOTFOUND (because we had deleted it), but the ACTION is Y which means do apply, so therefore we do the UPDATE. The ACTION is Y because the CONFLICT ACTION for the Q subscription HMTAB0002 is F.

From CLP-B, running the preceding query results in:

```
EXCEPTION_TIME          SUBNAME    REASON    SQLCODE   SQLSTATE OP  >
2007-04-08-16.54.32.812000 HMTAB0001 CHECKFAILED 100 02000 DELETE   >

A CR SRC_TRANS_TIME
N A  2007-04-08-15.54.28.000050
```

What the preceding command tells us is that Q subscription HMTAB0001 tried to DELETE a record (on DB2B), but we encountered a CHECKFAILED, which means …the conflict detection rule was to check all values or check changed values, and a nonkey value was not as expected. The ACTION is no apply, so we do not do anything (apart from log the exception).

We can see that on DB2A we trapped the UPDATE command and on DB2B we trapped the DELETE command.

Note also that the SRC_TRANS_TIME value is the same on DB2A and DB2B—this is the way to relate the two exceptions.

From the SELECT statement on both tables we saw that the UPDATE was honored and the DELETE wasn't. So why has this occurred?

On DB2A, the CONFLICT_ACTION for the target table is I–which means that Q Apply on DB2B will not force the change that it received from DB2A which is the DELETE command. On DB2B, the CONFLICT_ACTION for the target table is F—which means that Q Apply on DB2A will force the change that it received from DB2B, which is the UPDATE command.

The preceding setup makes DB2B the master copy and DB2A the slave copy. Changes on DB2B will always take precedence over changes made to the same row on DB2A.

Let's look at the contents of the IBMQREP_EXCEPTION table in more detail. We can see what caused the exception by looking in the TEXT column. The query we would use is:

```
$ db2 "select is_applied, substr(text,1,1000) from asn.ibmqrep_
exceptions"
```

From CLP-A, running the preceding query results in:

```
Y           UPDATE "ERIC"."HMTAB" SET "DEPT" = 'XXX  ' WHERE "ID" = 2
AND "DEPT" = 'Trn  ' AND "NAME" = 'Helen     '
```

We can see that the exception was applied (the first Y) and we can see what was applied.

From CLP-B, running the preceding query results in:

```
N           DELETE FROM "FRED"."HMTAB" WHERE "ID" = 2 AND "DEPT" =
'Trn  ' AND "NAME" = 'Helen     '
```

We can see that the exception was NOT applied (the first N) and what was rejected.

We have now finished the main part of the scenario.

We have seen that the choice of **which server should take precedence if a conflict is detected** when defining a Q Subscription for a table determines which database is the master and which is the slave.

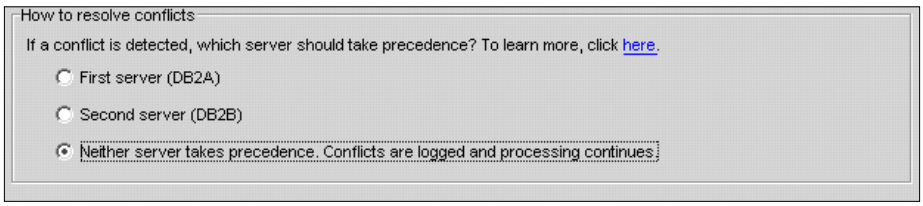
In this scenario, if server A fails we would ONLY switch the application to server B, when all the transactions from server A have been processed on server B. To determine when this point has been reached, we would need to monitor the Receive Queue on server B.

```
> runmqsc QMB
: dis ql(CAPA.TO.APPB.RECVQ) curdepth
```

Only when the current depth was zero would all the transactions from server A have been processed and we could now connect the application to server B.

Scenario 2—Bidirectional no master

Now let's look at how to make neither database, the master. From the Replication Center we have to specify that **neither server takes precedence** when we set up the Q subscription:



If we are using ASNCLP commands, then the `conflict action` should be set to `I` for both nodes. For the test, we will use a table called `HMTAB3` instead of `HMTAB`, but structured the same as `HMTAB`.

If we create the table and Q subscription. From CLP-A, issue:

```
$ db2 "create table eric.hmtab3 (id int not null, name char(10), dept
char(5)) "
$ db2 "alter table eric.hmtab3 add primary key (id) "
$ db2 "insert into eric.hmtab3 values(1,'Heather','Ops') "
```

And create a Q subscription with the `conflict action` set to `I` for both nodes, then if we check the `CONFLICT_ACTION` column of the `IBMQREP_TARGETS` table on each system using the following query, we should see the answers displayed:

From CLP-A, issue:

```
$ asnclp -f SYSA_loaddelupd3.asnclp
$ db2 "select substr(subname,1,10) as subname, conflict_action from asn.
ibmqrep_targets"
SUBNAME    CONFLICT_ACTION
---------- ---------------
HMTAB30001 I
```

On `DB2A`, we can see that the `CONFLICT_ACTION` value is `I`.

From CLP-B, issue the preceding query:

```
SUBNAME    CONFLICT_ACTION
---------- ---------------
HMTAB30002 I
```

On DB2B, we can see that the CONFLICT_ACTION value is I.

We can see that the CONFLICT_ACTION values for HMTAB3 have changed from the settings for table HMTAB—they are now both I.

Now if we run conf30, we get:

conf300	
conf301	**conf302**
conf01 - delete	conf02 - update
$ db2 connect to DB2A	**$ db2 connect to DB2B**
$ db2 select current timestamp from sysibm.sysdummy1 with ur	**$ db2 select current timestamp from sysibm.sysdummy1 with ur**
1 --- 2005-01-03-19.28.43.484000	1 --- 2005-01-03-19.28.43.675002
$ db2 delete from eric.hmtab where id = 2	**$ db2 update fred.hmtab set dept = 'XXX' where id = 2**
DB20000I The SQL command completed successfully.	DB20000I The SQL command completed successfully.
$ db2 select current timestamp from sysibm.sysdummy1 with ur	**$ db2 select current timestamp from sysibm.sysdummy1 with ur**
1 --- 2005-01-03-19.28.43.624000.	1 --- 2005-01-03-19.28.43.775000

We can check that the DELETE and UPDATE have worked as per the preceding output by selecting from both databases.

From CLP-A, issue:

```
$ db2 "select * from eric.hmtab3"
```

```
ID          NAME       DEPT
----------- ---------- -----
          1 Heather    Ops
          3 Chantal    Mgm
```

We can see that the `DELETE` has been processed for record `2`.

From CLP-B, issue:

```
$ db2 "select * from fred.hmtab3"
ID          NAME       DEPT
----------- ---------- -----
          1 Heather    Ops
          2 Helen      XXX
          3 Chantal    Mgm
```

We can see that the `UPDATE` has been processed for record `2`.

Let's look at the contents of the `IBMQREP_EXCEPTIONS` table in more detail using the following query:

```
$ db2 "select exception_time, substr(subname,1,10) as subname, reason,
sqlcode, sqlstate, substr(operation,1,10) as op, is_applied as a,
conflict_rule as CR, src_trans_time from asn.ibmqrep_exceptions"
```

From CLP-A, running the preceding query results in:

```
EXCEPTION_TIME           SUBNAME    REASON     SQLCODE   SQLSTATE  .>
2004-12-31-14.22.28.893000 HMTAB0001  NOTFOUND       100 02000    . >
2005-01-02-13.16.01.164001 HMTAB20001 NOTFOUND       100 02000     .>
2005-01-03-19.28.46.348000 HMTAB30001 NOTFOUND       100 02000     .>

OP         A CONFLICT_RULE SRC_TRANS_TIME
UPDATE     Y -             2004-12-31-14.22.27.000001
UPDATE     N -             2005-01-02-13.16.00.000001
UPDATE     N -             2005-01-03-19.28.43.000001
```

From CLP-A, running the preceding query results in:

```
EXCEPTION_TIME          SUBNAME    REASON       SQLCODE     SQLSTATE
>
2004-12-31-14.22.30.806000 HMTAB0002  CHECKFAILED      100 02000    >
2005-01-02-13.16.03.718000 HMTAB20002 CHECKFAILED      100 02000    >
2005-01-03-19.28.44.646000 HMTAB30002 CHECKFAILED      100 02000    >
```

```
OP          A CONFLICT_RULE SRC_TRANS_TIME
DELETE      N -             2004-12-31-14.22.27.000001
DELETE      Y -             2005-01-02-13.16.00.000001
DELETE      N -             2005-01-03-19.28.43.000001
```

Secondly, use the following query:

```
$ db2 "select is_applied, substr(text,1,1000) from asn.ibmqrep_
exceptions"
```

From CLP-A the above command returns:

```
N          <?xml version="1.0" encoding="ibm-1252_P100-2000" ?><msg
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespac
eSchemaLocation="mqcap.xsd" version="1.0.0" dbName="DB2A"><updateRow
subName="HMTAB20001"><col name="ID" isKey="1"><integer><afterVal>2</
afterVal></integer></col><col name="DEPT"><char><beforeVal>Trn  </
beforeVal><afterVal>XXX  </afterVal></char></col><col name="NAME"><ch
ar><afterVal>Helen     </afterVal></char></col></updateRow></msg>
```

We can see that the exception was NOT applied (the N at the beginning of the line) and we can see what was not applied.

From CLP-B, the preceding command returns:

```
N          <?xml version="1.0" encoding="ibm-1252_P100-2000"
?><msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocatio

n="mqcap.xsd" version="1.0.0" dbName="DB2B"><deleteRow
subName="HMTAB30002"><col name="ID" isKey="1"><integer>2</integer></
col></deleteRow></msg>
```

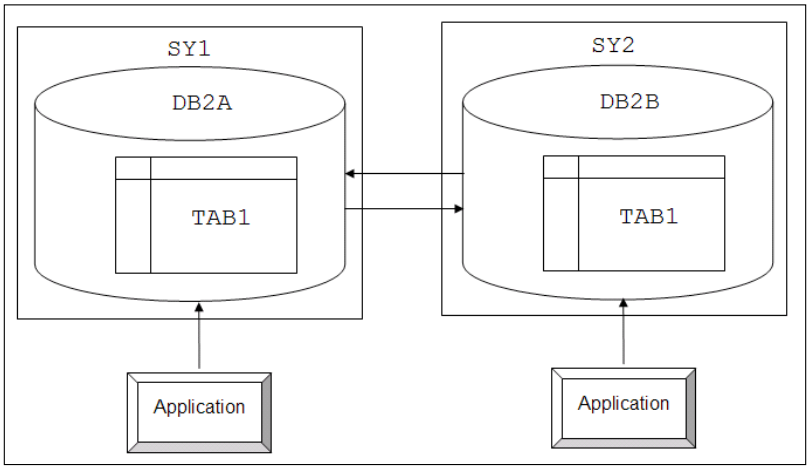
We can see that the exception was not applied (the `N` at the beginning of the line) and that what was not applied was a `DELETE` command.

We can see that neither `DB2A` or `DB2B` are the master database—each database will apply the updates to its own system. If there are any conflicts, then these are logged and not applied, but neither side wins. This could lead to the contents of the two databases diverging.

We have now finished looking at what happens if we do not make either `DB2A` or `DB2B` the master database.

Scenario 3—Bidirectional active/active

In a bidirectional scenario, we need to evaluate whether conflicts are possible from an application point of view. Consider an expanded diagram of Scenario 2. We have bidirectional replication set up between two systems (`SY1` and `SY2`):



We do not want an active/passive setup, but one where our application will update the same table on both systems, but at any one time each database must have a complete set of records from both systems. To achieve this, the application has been altered so that users attached thru the application to `SY1` will process the odd rows of `TAB1` and users attached thru the application to `SY2` will process the even rows of `TAB1`.

This scenario will handle all the standard failure scenarios that Q replication can handle (that is, server `SY1` becomes unavailable and then becomes available again). One failure scenario that we have to be aware of is what happens if the Q subscription for `TAB1` becomes inactive for whatever reason (perhaps someone has mistakenly inactivated it)—what happens then? The applications on `SY1` and `SY2` will still insert/update/delete their respective rows in `TAB1`, but these operations will NOT get replicated to the other system, which means that the table contents will diverge, which is clearly bad news! To rectify this problem is not a trivial matter. We cannot simply activate the Q subscription again, as the table contents will be different on both sides.

What we have to do is as follows:

Switch the application from `SY2` to `SY1` so that all applications are running against `SY1`. This will ensure that no more updates will occur on `TAB1` on `SY2`. Let's call this point in time timestamp2.

Find the timestamp of when the Q subscription became inactive. Let's call this point in time timestamp1.

We then need to get all the operations that occurred against `TAB1` on `SYS2` between timestamp1 and timestamp2. We can use the log analysis function of the IBM Recovery Expert product to achieve this.

We can then replay these transactions against `TAB1` on `SY1` so that this table will now have a complete set of records.

Once `TAB1` on `SY1` contains a complete set of records, then we can reactivate the Q subscription and perform a full refresh of `TAB1` on `SY2`.

When the full refresh has completed, we can switch the applications we moved from `SY2` earlier back to `SY2`.

This recovery process will only work if we can perform the log analysis part. If we do not have a log analysis product, then this type of setup is best avoided.

Summary

In this book, we started by looking at the tools available to set up Q replication, and then moved on to look at the pre-setup evaluation that you need to do to ensure that the replication solution is the appropriate one. We covered 12 scenarios ranging from the simplest unidirectional replication to P2P four-way, also covering Data Event Publishing and using a stored procedure to transform data. We have given you all the commands you need to set up any of these scenarios. We looked at conflict detection and gave some worked through examples. We also looked at the important example of having an active/active setup and that we need some sort of log analysis tool before attempting such a setup.